(2)

WRDC-TR-89-3125

Volume III

MINIMUM FLYING QUALITIES

Volume III: Program CC's Implementation of the Human Optimal Control Model

Peter M. Thompson

Systems Technology, Inc.
13766 South Hawthorne Blvd
Hawthorne, CA 90250-7083

January 1990

Final Report for Period October 1985 - July 1989

FLIGHT DYNAMICS LABORATORY
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6553

[illegible]
CAPT MARK J. DETROIT, USAF
Control Dynamics Branch
Flight Control Division

[illegible]
DAVID K. BOWSER, Chief
Controls Dynamics Branch
Flight Control Division

FOR THE COMMANDER

[illegible]
H. MAX DAVIS, Assistant for
Research and Technology
Flight Control Division
Flight Dynamics Laboratory

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
STI-TR-1235-1-III

5. MONITORING ORGANIZATION REPORT NUMBER(S)
WRDC-TR-89-3125, Volume III

6a. NAME OF PERFORMING ORGANIZATION
Systems Technology, Inc.

6b. OFFICE SYMBOL (If applicable)

7a. NAME OF MONITORING ORGANIZATION
Flight Dynamics Laboratory (WRDC/FIGCB)
Wright Research and Development Center

6c. ADDRESS (City, State, and ZIP Code)
13766 South Hawthorne Boulevard
Hawthorne, California 90250-7083

7b. ADDRESS (City, State, and ZIP Code)
Wright-Patterson Air Force Base, Oh 45433-6553

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
Flight Dynamics Laboratory
Wright Research and Development Center

8b. OFFICE SYMBOL (If applicable)
WRDC/FIGCB

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F33615-85-C-3610

8c. ADDRESS (City, State, and ZIP Code)
Wright-Patterson AFB OH 45433-6553

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO
62201F	2403	05	64

11. TITLE (Include Security Classification)
Minimum Flying Qualities
Volume III: Program CC's Implementation of the Human Optimal Control Model

12. PERSONAL AUTHOR(S)
Peter M. Thompson

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM Oct. '85 TO Jul. '89

14. DATE OF REPORT (Year, Month, Day)
1990 January

15. PAGE COUNT
88

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
01	03	
17	07	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Flying Qualities
Minimum Flying Qualities
Multi-Axis Flying Qualities
Pilot Modeling
Optimal Control Pilot Modeling
Piloted Simulation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The project was initiated to explore the modern nature of minimum flying qualities in the presence of modern aircraft and multi-redundant flight control system technology. It had several phases, including: 1) an intensive effort to develop and/or elaborate existing pilot modeling analysis techniques to apply to situations associated with minimum flying qualities, divided attention pilot operations, and multi-axis control tasks; 2) preliminary analyses and associated fixed base simulations to expand the meager multi-axis data base and to serve as pilot studies for more extensive simulations on the Air Force's Large Amplitude Multimode Aerospace Research Simulator. 3) an extensive simulation program on LAMARS to investigate minimum flying qualities and related situations; and 4) analysis and interpretation of both the early and LAMARS simulation efforts in the context of the pilot modeling advances. The project documentation appears in three volumes. Volume I reports on the results of 2) through 4) above. Volume II is a stand-alone monograph on pilot modeling, including procedures for estimating pilot workload as "measured" by pilot ratings. ~~Volume III~~ is a stand-alone monograph which presents a detailed implementation of a much expanded version of the human optimal control model on Program CC.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
[✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL
Capt Mark J. Detroit

22b. TELEPHONE (Include Area Code)
(513) 255-8499

22c. OFFICE SYMBOL
WRDC/FIGC

DD Form 1473, JUN 86 Previous editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Contents

1 INTRODUCTION 3

2 TECHNICAL PROBLEM STATEMENT 5
- 2.1 Controlled System ... 5
- 2.2 Performance Index ... 8
- 2.3 LQR Solution ... 8
- 2.4 KBF and Linear Predictor Solutions ... 10
- 2.5 Noise Ratios, Indifference Thresholds, and Fractional Attentions ... 11
- 2.6 Performance Measures ... 14
- 2.7 Frequency Responses ... 17

3 OCM IMPLEMENTATION 22
- 3.1 Overview ... 22
- 3.2 Using Program CC ... 26
  - 3.2.1 Setup ... 26
  - 3.2.2 Executing Macros ... 27
  - 3.2.3 User-Defined-Commands ... 28
  - 3.2.4 Example ... 28
- 3.3 The Controlled System ... 29
- 3.4 The Optimal Solution ... 29
- 3.5 Analyzing the Results ... 32
- 3.6 Computation Time ... 34

4 EXAMPLES 36
- 4.1 Integrator ... 36
- 4.2 Double Integrator ... 42
- 4.3 Single Axis Dander Example ... 46
- 4.4 Multi-Axis Dander Example ... 47

A How to Use Program CC, Version 4 53
- A.1 The Basics ... 53
- A.2 Data Types ... 55
- A.3 The CC Command Level ... 56
- A.4 The STATE Command Level ... 58
- A.5 GRAPHICS ... 64
- A.6 The DATA Command Level ... 69
- A.7 Making Macros ... 69

B Macro Listings 73

LIST OF FIGURES 2

1. The Optimal Control Model 6

2. Computational Flow for the OCM 7

3. Frequency Domain Version of OCM 18

4. Different Configurations for $(Y_c)_{all}$ 30

5. Summary of the Integral OCM Example 39

6. Frequency Responses for Integral OCM Example 41

7. Summary of Double Integrator OCM Example 44

8. Frequency Responses for Double Integrator OCM Example 45

9. Summary of Dander Single Axis Example 48

10. Frequency Responses for Dander Single Axis Example 49

11. Attentional Fraction Survey for Dander Multi-Axis Example 51

12. Program CC Plot Including the Plot Option Blocks 66

LIST OF TABLES 2

1. OCM Macros 23

2. Input Parameters 24

3. Output Parameters, Single-Input System 25

4. KBF Iterations 37

1 INTRODUCTION

The optimal control model (OCM) is based on the assumption that a human operator estimates the state of the controlled system and develops a control strategy which minimizes a performance index. The pioneering work of Kleinman, Baron, and Levison [1], of Bolt, Beranek, and Newman (BBN) used this basic assumption to set up an optimal control problem which closely agrees with experimental tracking data. The resulting controller consists of a Kalman Bucy Filter (KBF), a linear predictor, and a set of Linear Quadratic Regulator (LQR) gains.

Practical understanding of the OCM develops through use, which in turn requires computer implementation. Though implementations exist, notably PIREP [2], they are not readily available. The purpose of this report is to describe an implementation of the OCM using Program CC [3]. This will greatly increase the availability and understanding of the OCM, both here at STI and elsewhere.

The OCM has changed, but not dramatically, since its introduction in the late 1960's. The seminal reference [1], see also [4,5], has the following features: a performance index which uses a weighted sum of mean square error and control rate energy, full attention noise ratios for observation and motor noises, and an iterative solution method to achieve the desired noise ratios. None of this has changed.

Additional features and numerous applications have appeared in the years following the OCMs introduction, e.g. [6]–[10]. The additional features included in Program CC's implementation are visual indifference thresholds and fractional attention parameters. Notably absent is the use of pseudo-noise to induce low frequency phase droop, and the optimization of fractional attention for multi-input problems.

The current interest is to understand and predict human operator behavior in multi-axis tasks and in divided attention situations. In particular, the objective is to predict pilot behavior and ratings for multi-axis tasks, and to compare these predictions against experimental data. Previous work [11]–[14] has suggested that the value of the optimal control performance index correlates with pilot opinion ratings (POR's) such as the Cooper-Harper scale.

The motivation for implementing the OCM on Program CC is to further develop the ability to predict POR's. A secondary objective is to simplify the implementation and broaden the availability of the OCM. This report describes the implementation of the OCM. Volumes 1 and 2 detail some of the applications.

The implementation consists of several Program CC macros and one user defined command. The macros will only work with Version 4 of Program CC. The macros are used in sequence to (1) create a state space model for Y_c and Y_w, the controlled element and driving noise filter, (2) iterate the LQR problem in order to set the neuro-muscular mode or modes, (3) set up the KBF/linear predictor problems. (4) iterate the KBF problem until desired noise ratios are achieved, and (5) analyze the resulting Y_p pilot model. The user defined command is called from within the macros as part of the LQR and KBF iterations. In addition to the macros, all of the existing and powerful capabilities in Program CC can be used for analyzing the resulting system models and pilot models.

Section 2 contains a technical description of the OCM. It is presented in its general multivariable form, but only the single-input version is implemented. The description brings together material scattered among several references [1,2],[4]-[9], and adds a new twist to the treatment of delay which results in state space and transfer function models for the pilot. Section 3 is the main reference for the Program CC implementation; providing usage notes, and listing in Tables 1-3 the macro names, macro parameters, and data storage locations. Several examples are presented in Section 4, including the use of the optimal cost to predict pilot opinion ratings. Background information on Program CC is provided in Appendix A, and the macros are listed in Appendix B.

Fortunately it is not necessary to completely understand the technical details of the OCM in order to use it. It is sufficient to use the OCM as a means to an end: a long handled crank which results in a Y_p. Assuming some prior knowledge about the OCM and about Program CC, the minimum amount of addition information needed to operate the macros is contained in Tables 1 and 2.

2 TECHNICAL PROBLEM STATEMENT

A complete technical description of the optimal control model is presented in this section. The description is valid for the multivariable case, though only the single–input single–output case is implemented. The equations are summarized in the block diagram of Figure 1, and the computational flow is summarized in Figure 2.

2.1 Controlled System

The controlled system is modeled using state space equations. The driving noise $w(t)$ will typically contain dynamics, which are included together with the system dynamics in the A matrix. The only difference between the single-input and multi-input versions is the dimension of the input vectors.

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) + Ew(t)\\ y(t) &= Cx(t) + Du(t)\\ y_p(t) &= y(t-\tau) + v_y(t-\tau)\end{aligned}$$

The dimensions and definitions of the vectors are as follows. The notation R^n indicates a vector of n real numbers.

$x(t) \in \mathrm{R}^n$	state	$y_p(t) \in \mathrm{R}^r$	observation
$u(t) \in \mathrm{R}^m$	input	$w(t) \in \mathrm{R}^{m_e}$	driving noise
$y(t) \in \mathrm{R}^r$	output	$v_y(t) \in \mathrm{R}^r$	observation noise

The noise intensities are:

$$\begin{aligned}\mathbf{E}\{w(t)w(t-\sigma)\} &= V_u\delta(\sigma)\\ \mathbf{E}\{v_y(t)v_y(t-\sigma)\} &= V_y\delta(\sigma)\end{aligned}$$

It is assumed that the human uses both errors and error rates, and therefore the output $y(t)$ contains both. It is not important how they the outputs are arranged, so for notational convenience the errors are grouped together and listed first:

$$y(t) = \begin{pmatrix} y_e(t) \\ \dot{y}_e(t) \end{pmatrix}$$

The transfer function $Y_c(s)$ for the controlled system is defined to be from the input u to the error y_e, because the single-input case this is the traditional definition. The Program CC macros described in Section 3 use variations of $Y_c(s)$ with different inputs and outputs.

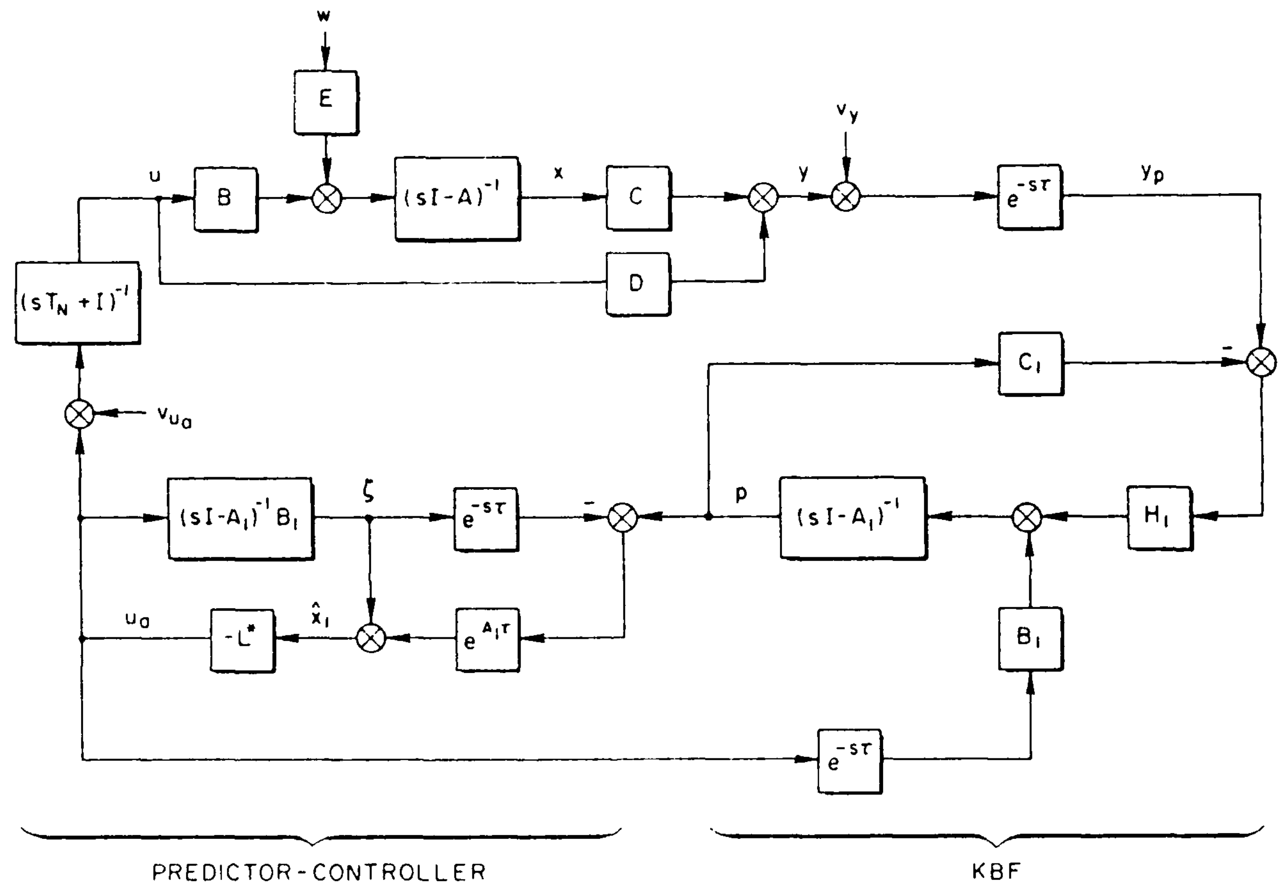


Figure 1: The Optimal Control Model

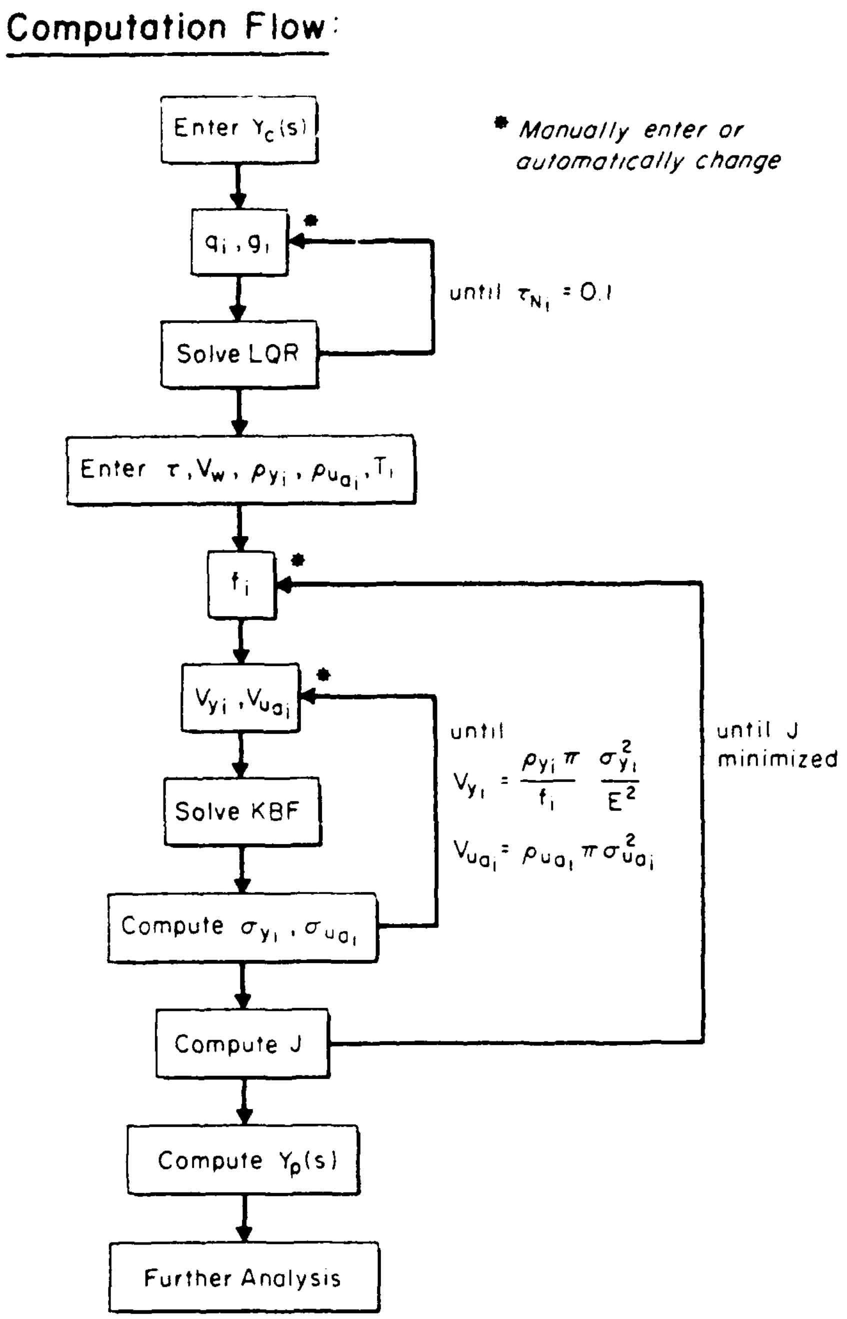

Computation Flow:
Enter $Y_c(s)$
q_i, g_i
* Manually enter or automatically change
until $\tau_{N_i} = 0.1$
Solve LQR
Enter $\tau, V_w, \rho_{y_i}, \rho_{u_{a_i}}, T_i$
f_i
$V_{y_i}, V_{u_{a_i}}$
until $V_{y_i} = \frac{\rho_{y_i}\pi}{f_i}\frac{\sigma^2_{y_i}}{E^2}$ $V_{u_{a_i}} = \rho_{u_{a_i}}\pi\sigma^2_{u_{a_i}}$
until J minimized
Solve KBF
Compute $\sigma_{y_i}, \sigma_{u_{a_i}}$
Compute J
Compute $Y_p(s)$
Further Analysis

Figure 2: Computational Flow for the OCM

The human visual delay of τ seconds is included in the definition of the observation $y_p(t)$. Typically for the OCM $\tau = .2$ is used. Note that this τ is not the same value as the effective delay, τ_e, which is based on the unit magnitude crossover of Y_pY_c. Experimentally determined values for τ_e range from .15 to .25 seconds and depends on the amount of pilot lead.

The driving noise intensity V_w is determined as part of the experimental setup. The observation noise intensity V_y is automatically determined according to an implicit equation explained in Section 2.5.

2.2 Performance Index

The objective of the OCM is to determine a control law $u(t) = f(y_p(t))$ which minimizes the performance index:

$$J = \lim_{T\to\infty} \mathbf{E}\left\{\frac{1}{T}\int_0^T x'Qx + \dot{u}'G\dot{u})\,dt\right\}$$

In most if not all of the cases in which the OCM has been used, the weighting matrices take the diagonal form:

$$\begin{aligned} Q &= C'Q_yC \\ Q_y &= \mathrm{diag}\{q_1,\ldots,q_r\} \\ G &= \mathrm{diag}\{g_1,\ldots,g_m\} \end{aligned}$$

The q_i's corresponding to the errors are scaled to give approximate equal weight to standard deviations, and the q_i's corresponding to error rates are set to zero:

$$q_i = \begin{cases} 1/\sigma^2_{y_i} & i = 1,2,\ldots,r/2 \\ 0 & i = r/2+1,\ldots,r \end{cases}$$

The g_i's are chosen to in order to place the *neuro-muscular* dynamics, as explained in the next subsection.

The OCM differs from the standard Linear Quadratic Gaussian (LQG) optimal control problem in three respects: (1) the weighting on control rate, (2) the inclusion of motor noise before the neuro-muscular dynamics, and (3) the observation delay of τ seconds.

2.3 LQR Solution

The stochastic Linear Quadratic Regulator problem is solved to determine the full state control weights. The system is augmented with a preceding integrator, which results in an equivalent performance index without a control rate.

$$\begin{aligned}\dot{x}_0(t) &= A_0 x_0(t) + B_0\mu + E_0 w \\ J &= \lim_{T\to\infty} \mathbf{E}\left\{\frac{1}{T}\int_0^T (x_0' Q_0 x_0 + \mu' G \mu)\, dt\right\}\end{aligned}$$

where:

$$\begin{aligned} A_0 &= \begin{pmatrix} A & B \\ 0 & 0 \end{pmatrix} \qquad B_0 = \begin{pmatrix} 0 \\ I \end{pmatrix} \qquad E_0 = \begin{pmatrix} E \\ 0 \end{pmatrix} \\ Q_0 &= \begin{pmatrix} Q & 0 \\ 0 & 0 \end{pmatrix} \qquad x_0 = \begin{pmatrix} x \\ u \end{pmatrix} \qquad \mu = \dot{u}\end{aligned}$$

The solution is:

$$\begin{aligned}\mu &= -Lx_0 \\ L &= G^{-1} B_0' K_0\end{aligned}$$

where K_0 is the unique positive semi-definite solution of the algebraic Riccati equation (ARE):

$$0 = A_0' K_0 + K_0 A_0 + Q_0 - K_0 B_0 G^{-1} B_0' K_0$$

The solution method of choice for the ARE is to (1) set up the Hamiltonian matrix, (2) compute its Schur decomposition, (3) order the eigenvalues of the Hamiltonian so the stable eigenvalues are in the upper left hand block T_{11} of upper triangular Schur matrix T, and then (4) compute K_0 using the partitioned Schur vectors U which span the stable subspace. This is a modified Potter's method, due to Laub. Schur vectors are computed as an intermediate step towards eigenvectors, the latter of which are in general numerically ill-conditioned.

$$\begin{aligned}\begin{pmatrix} A_0 & -B_0 G^{-1} B_0' \\ -Q_0 & -A_0' \end{pmatrix} &= \begin{pmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{pmatrix} \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{pmatrix}' \\ K_0 &= U_{21} U_{11}^{-1}\end{aligned}$$

The control gains can be partitioned to separately apply to the state x and the input u:

$$\mu - \begin{pmatrix} L_1 & L_2 \end{pmatrix} \begin{pmatrix} x \\ u \end{pmatrix}$$

By convention the feedback weight on u is closed prior to the other feedbacks resulting in the neuro-muscular dynamics. Input to the neuro-muscular dynamics then consists of the remaining feedbacks plus injected motor noise.

$$\begin{aligned} T_N \dot{u} &+ u = u_a + v_{u_a} \\ u_a &= -L^* x_0 \end{aligned}$$

where:

$$\begin{aligned} T_N &= L_2^{-1} \\ L^* &= (\, L_2^{-1} L_1 \quad 0 \,) \end{aligned}$$

and where the motor noise intensity is:

$$\mathbf{E}\{v_{u_a}(t) v_{u_a}(t-\sigma)\} = V_{u_a} \delta(\sigma)$$

The value of V_{u_a} is automatically determined as described in Section 2.5.

The input rate weight G is adjusted so as to achieve neuro- muscular dynamics of 10 rad/sec. In the single-input case a simple binary search yields the correct value of G. In the case of decoupled inputs separate binary searches can be used for each input. For cases more complicated no help is available.

2.4 KBF and Linear Predictor Solutions

The states are not available for feedback, only the delayed and noise corrupted observation $y_p(t)$. The optimal solution proceeds by using a KBF to estimate the state at time $t - \tau$, and then using a linear predictor to estimate the state at time t. Due to the convention used for the neuro-muscular dynamics, the following augmented system is used for the KBF:

$$\begin{aligned} \dot{x}_1(t) &= A_1 x_1(t) + B_1 u_a + w_1 \\ y_p(t) &= C_1 x_1(t-\tau) + v_y(t-\tau) \end{aligned}$$

where:

$$A_1 = \begin{pmatrix} A & B \\ 0 & -L_2 \end{pmatrix} \qquad B_1 = \begin{pmatrix} 0 \\ L_2 \end{pmatrix}$$

$$x_1 = \begin{pmatrix} x \\ u \end{pmatrix} \qquad w_1 = \begin{pmatrix} w \\ v_u \end{pmatrix} \qquad C_1 = (\, C \quad 0 \,)$$

$$\mathbf{E}\{w_1(t) w_1(t-\tau)\} = W_1 = \begin{pmatrix} E V_u E' & 0 \\ 0 & L_2 V_{u_a} L_2' \end{pmatrix}$$

The KBF computes $p(t)$ the linear mean-square estimate of $x_1(t-\tau)$ based on observations $y_p(\sigma)$ for $\sigma \leq t$:

$$\dot{p}(t) = A_1 p(t) + H_1 [y_p(t) - C_1 p(t)] + B_1 u_a(t)$$

where the filter gain is:

$$H_1 = \Sigma_1 C_1' V_y^{-1}$$

and where Σ_1 is the unique positive semi-definite solution of the ARE:

$$0 = A_1\Sigma_1 + \Sigma_1 A_1' + W_1 - \Sigma_1 C_1' V_y^{-1} C_1 \Sigma_1$$

The linear predictor updates $p(t)$ to obtain $\hat{x}_1(t)$, the linear mean-square estimate of $x_1(t)$ based on observations $y_p(\sigma)$ for $\sigma \leq t$. Note that $p(t-\tau) \neq \hat{x}_1(t)$.

$$\begin{aligned} \dot{\xi}(t) &= A_1\xi(t) + B_1 u_a(t) \\ \hat{x}_1 &= \xi(t) + e^{A_1\tau}[p(t) - \xi(t-\tau)] \end{aligned}$$

The LQR control weights are applied to $\hat{x}_1(t)$, again used according to the convention for the neuro-muscular dynamics:

$$u_a(t) = -L^* \hat{x}_1(t)$$

2.5 Noise Ratios, Indifference Thresholds, and Fractional Attentions

The observation noise intensity V_y and the motor noise intensity V_{u_a} are assumed to be diagonal matrices:

$$\begin{aligned} V_y &= \mathrm{diag}\{v_{y_1}, \ldots, v_{y_r}\} \\ V_{u_a} &= \mathrm{diag}\{v_{u_{a_1}}, \ldots, v_{u_{a_m}}\} \end{aligned}$$

where each of the diagonal elements satisfy:

$$v_{y_i} = \frac{\rho_{y_i}\pi}{f_i} \frac{\sigma_{y_i}^2}{E(\sigma_{y_i}, T_i)^2} \quad i = 1, \ldots, r \tag{1}$$

$$v_{u_{a_i}} = \rho_{u_{a_i}} \pi \sigma_{u_{a_i}}^2 \quad i = 1, \ldots, m \tag{2}$$

and where the individual terms in (1.2) are defined:

$\rho_{y_i}, \rho_{u_{a_i}}$	$=$	noise ratios
$\sigma_{y_i}, \sigma_{u_{a_i}}$	$=$	standard deviations of y_i, u_{a_i}
f_i	$=$	fraction attentions
T_i	$=$	indifference thresholds
$E(\sigma_{y_i}, T_i)$	$=$	$\mathrm{erfc}\left(\dfrac{T_i}{\sigma_{y_i}\sqrt{2}}\right)$

Equations (1.2) for the noise intensities v_{y_i}'s and $v_{u_{a_i}}$'s are implicit, because the standard deviations σ_{y_i}'s and $\sigma_{u_{a_i}}$'s depend on the noise intensities. An iteration is required in order to compute the noise intensities. The iteration takes the following form: (1) guess initial values for the noise intensities, (2) compute the KBF gain H_i, followed by the standard deviations, (3) if equations (1.2) are satisfied within a specified tolerance then stop, else use equations (1.2) to compute new values for the v_{y_i}'s and $v_{u_{a_i}}$'s, and then repeat.

The *observation noise ratios* ρ_{y_i} have been experimentally determined to be .01. Usually this ratio is expressed using power dB $10\log_{10}$, or in this case -20 dB.

The *motor noise ratios* can be measured in principle, but there has not been any found in practice (when dither, stick pumping, etc is excluded). Various hypotheses have been proposed for the origin of motor noise, but it is more likely that motor noise is a mathematical fiction used to achieve a solution. Numbers used for the $\rho_{u_{a_i}}$'s vary from .001 to .05, some researchers preferring the smaller values and some the larger. A "safe" value seems to be .01.

The output y_i must move past an *indifference threshold* T_i before a change is perceived. This threshold is modeled by a Gaussian describing function. The input of the threshold at a particular time is normally distributed with mean zero and standard deviation σ_{y_i}, and the output is normally distributed with mean zero and standard deviation $\sigma_{y_i}/E(\sigma_{y_i}, T_i)$, where:

$$E(\sigma_{y_i}, T_i) = \mathrm{erfc}\left(\frac{T_i}{\sigma_{y_i}\sqrt{2}}\right) = \mathrm{Prob}(|x| > T_i)$$

where x is a sample from a normal distribution with mean zero and standard deviation σ_{y_i}.

When the indifference threshold is $T_i=0$ then $E=1$, and it is always the case that $0<E\leq 1$. Visual perception thresholds have experimentally been determined to be .05 deg and .1 deg/sec. The value of T_i used in the OCM depends on how far the display is from the human.

The observation noise is assumed to be inversely proportion to the *fractional attention* f_i, e.g. the observation noise intensity v_{y_i} doubles if only 50% attention is being paid to output y_i. It does not matter how the remaining attention is allocated, whether for other control tasks or for non-control tasks such as communication. The usual assumption is that the same attention is attributed to the error and error rate in a given axis, therefore:

$$\sum_{\textbf{axis}} f_i = 1$$

Or in terms of the outputs:

$$\sum_{i=1}^{r} f_i = 2$$

Some researchers allow f_i to differ for error and error rate, and/or require the total across outputs to be 1. In the single-input (single-axis) case simply use $f_1=f_2=1$. In the multi-axis case, if the tasks are of comparable difficulty then apply equal attention to each axis, otherwise optimize over the f_i so as to minimize the performance index J. Program CC allows arbitrary input of f_i and does not check the sum, but does not optimize over f_i.

Pseudo motor noise is a scheme where large values of $\rho_{u_{a_i}}$ are used to compute the KBF gains, and then smaller values are used to compute the standard deviations σ_{y_i}'s and $\sigma_{u_{a_i}}$'s. Different equations then those presented in Section 2.5 are needed in order the compute the standard deviations, due to the fact that the state estimation errors are no longer uncorrelated with the state estimates. The use of pseudo motor noise has been proposed to account for phase droop and motion cues. Pseudo motor noise is *not*, however, included as part of the Program CC implementation, in other words the same values for $\rho_{u_{a_i}}$ must be used for all calculations. This is no big loss. Pseudo motor noise is a historical oddity for which no general guidelines were developed and which never lived up to its expectations.

The π in equations (1,2) was the source of confusion to the author. Even though others are not likely to be similarly fooled, I'll take the liberty to explain. Neglect for this argument the attentional fraction and the indifference threshold, leaving:

$$V_y = \rho_y \pi \sigma_y^2$$

The mean square value of the output is defined using the autocorrelation function:

$$\sigma_y^2 = R_y(0)$$

where the autocorrelation is defined:

$$R_y(\tau) = \mathbf{E}\{y(t)y(t-\tau)\}$$

and where the autocorrelation is related to the power spectral density using the Fourier transform identities:

$$\begin{aligned} S_y(j\omega) &= \int_{-\infty}^{\infty} R_y(\tau)e^{-j\omega\tau}d\tau \\ R_y(\tau) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} S_y(j\omega)e^{j\omega\tau}d\omega \end{aligned}$$

The noise ratio ρ_y, in terms of the power spectral density is:

$$\rho_y = \frac{V_y}{\pi\sigma_y^2} = \frac{V_y}{\frac{1}{2}\int_{-\infty}^{\infty} S_y(j\omega)d\omega}$$

If, however, the integration of $S_y(j\omega)$ is defined *only* over the positive frequencies then the noise ratio is:

$$\rho_y = \frac{V_y}{\int_0^{\infty} S_y(j\omega)d\omega}$$

hence ρ_y can be defined as ratio of the input noise intensity to the positive frequency power density.

The OCM literature over the years defines the noise ratio ρ_y using positive frequencies (where no π is needed), but uses calculations in terms of σ_y^2 (where π is needed). To make a long story short, use π in the calculations.

2.6 Performance Measures

The time domain performance measures are the mean square errors and the optimal cost. The mean square for the augmented state is:

$$\begin{aligned} X &= \mathbf{E}\{x_1x_1'\} = e^{A_1\tau}\Sigma_1e^{A_1'\tau} + \int_0^{\tau} e^{A_1\sigma}W_1e^{A_1'\sigma}d\sigma \\ &+ \int_0^{\infty} e^{(A_1-B_1L^*)\sigma}e^{A_1\tau}H_1V_yH_1'e^{A_1'\tau}e^{(A_1-B_1L^*)'\sigma}d\sigma \end{aligned}$$

Define the estimation error $e_1 = x_1 - \hat{x}_1$. Then X can be decomposed into $X = \hat{E} + \hat{X}$, where:

$$\begin{aligned}
\hat{E} &= \mathbf{E}\{e_1 e_1'\} = e^{A_1\tau}\Sigma_1 e^{A_1'\tau} + \int_0^\tau e^{A_1\sigma} W_1 e^{A_1'\sigma} d\sigma \\
\hat{X} &= \mathbf{E}\{\hat{x}_1 \hat{x}_1'\} = \int_0^\infty e^{(A_1 - B_1 L^*)\sigma} e^{A_1\tau} H_1 V_y H_1' e^{A_1'\tau} e^{(A_1 - B_1 L^*)'\sigma} d\sigma
\end{aligned}$$

The augmented state is $x_1' = (\, x' \quad u' \,)$, hence the mean square values for x and u are respectively the X_{11} and X_{22} blocks. The mean square output and neuro-muscular input are:

$$\begin{aligned}
Y &= \mathbf{E}\{yy'\} = C_1 X C_1' \\
U_a &= \mathbf{E}\{u_a u_a'\} = L^* \hat{X} L^{*\prime}
\end{aligned}$$

The mean square of $\dot{u}$ is infinite because the white motor noise feeds directly into $\dot{u}$. Using instead the estimated value of $\dot{u}$:

$$U_{dot} = \mathbf{E}\{\dot{\hat{u}}\dot{\hat{u}}'\} = L\hat{X}L' + (\,0 \quad L_2\,)\hat{E}(\,0 \quad L_2\,)'$$

Finally, the optimal cost is:

$$\begin{aligned}
J &= \lim_{T\to\infty} \mathbf{E}\left\{ \frac{1}{T}\int_0^T (y'Q_y y + \dot{u}'G\dot{u})\, dt \right\} \\
&= \text{Trace}\left[\mathbf{E}\{yy'\}Q_y\right] + \text{Trace}\left[\mathbf{E}\{\dot{u}\dot{u}'\}G\right] \\
&= \text{Trace}[YQ_y] + \text{Trace}[U_{dot}G]
\end{aligned}$$

which in the single-input case is:

$$\begin{aligned}
J &= \lim_{T\to\infty} \mathbf{E}\left\{ \frac{1}{T}\int_0^T \left(qy_1^2 + g\dot{u}^2\right) dt \right\} \\
&= q\sigma_{y_1}^2 + g\sigma_{\dot{u}}^2
\end{aligned}$$

The types of calculations needed for these performance measures are matrix exponentials and Lyapunov equations. The preferred method for matrix exponentials is the scaled Pade approximation, (with the scaled Taylor series a close second and the unscaled Taylor series absolutely not to be used). The scaling is:

$$\hat{A} = A_1\tau/\alpha, \qquad \text{where } \alpha = 2^{2^m}$$

and where m is the smallest integer ≥ 0 such that α is greater than the square root of the sum of squares of the elements of $A_1\tau$. This scaling approximately keeps the eigenvalues of $A_1\tau$ less than 1. If $\hat{F} = \exp(\hat{A})$ and $F = \exp(A_1\tau)$, then rescaling is accomplished simply by $F = \hat{F}^{2^m}$, which requires only m matrix multiplications.

A block matrix is used to compute the integral of a matrix exponential:

$$\exp\begin{pmatrix} -A_1 & W_1 \\ 0 & A_1' \end{pmatrix}\tau = \begin{pmatrix} F_1 & G_1 \\ 0 & F_2 \end{pmatrix}$$

where:

$$\begin{aligned} F_1 &= e^{-A_1\tau} \\ F_2 &= e^{A_1'\tau} \end{aligned}$$

$$G_1 = \int_0^\tau e^{-A_1(\tau-\sigma)}W_1 e^{A_1'\sigma} d\sigma = e^{-A_1\tau}\int_0^\tau e^{A_1\sigma}W_1 e^{A_1'\sigma} d\sigma$$

Hence:

$$\hat{E} = e^{A_1\tau}\Sigma_1 e^{A_1'\tau} + \int_0^\tau e^{A_1\sigma}W_1 e^{A_1'\sigma} d\sigma = F_2'\Sigma_1 F_2 + F_2' G_1$$

The Lyapunov equation is:

$$0 = \hat{P}\hat{Q} + \hat{Q}\hat{P} + \hat{R}$$

where $\hat{R}$ is symmetric. The solution $\hat{Q}$ exists and is unique if the eigenvalues of $\hat{P}$ and $\hat{R}$ do not coincide. This condition is automatically satisfied if $\hat{P}$ is stable and $\hat{R} \geq 0$. If $\hat{P}$ is stable then the Lyapunov solution $\hat{Q}$ satisfies the integral identity:

$$\hat{Q} = \int_0^\infty e^{\hat{P}\sigma}\hat{R}e^{\hat{P}'\sigma} d\sigma$$

Hence the mean square value $\hat{X}$ is computed using the Lyapunov equation with:

$$\begin{aligned} \hat{P} &= A_1 - B_a L^* \\ \hat{R} &= e^{A_1\tau} H_1 V_y H_1' e^{A_1'\tau} \end{aligned}$$

The preferred solution method is Bartel-Stewart, using the Schur decomposition for $\hat{P}$.

2.7 Frequency Responses

The frequency responses of interest are $Y_c(s)$, $Y_p(s)$, and the remnant $\Phi_{nn_e}(s)$. Here it is explained how to compute the frequency response data, and a state space/transfer function approximation for $Y_p(s)$.

The pilot model $Y_p(s)$ is composed of a visual delay, KBF, linear predictor, and neuro-muscular mode.

$$\begin{aligned}
\dot{p} &= (A_1 - H_1C_1)p + H_1y(t-\tau) + B_1u_a(t-\tau) + H_1v_y(t-\tau) \\
\dot{\xi} &= A_1\xi + B_1u_a \\
\hat{x}_1 &= \xi(t) + e^{A_1\tau}[p(t) - \xi(t-\tau)] \\
u_a &= -L^*\hat{x}_1 \\
\dot{u} &= -L_2u + L_2u_a + L_2v_u
\end{aligned}$$

It is more convenient to replace the differential equation for ξ with another for $\hat{x}_1$:

$$\dot{\hat{x}}_1 = -e^{A_1\tau}H_1C_1p + (A_1 - B_1L^*)\hat{x}_1 + e^{A_1\tau}H_1y + e^{A_1\tau}H_1v_y$$

Move the visual delay from y over to u_a, which is equivalent from an input/output point of view. Combine p and $\hat{x}_1$ into an augmented vector x_2. Change the input matrices for y to obtain an equivalent system driven just by y_e. The result is:

$$\begin{aligned}
\dot{x}_2 &= A_2x_2 + \tilde{B}_2y_e + E_2u_a(t-\tau) + B_2v_y \\
u_a &= C_2x_2 + D_2y_e \\
\dot{u} &= -L_2u + L_2u_a(t-\tau) + L_2v_u
\end{aligned}$$

where:

$$A_2 = \begin{pmatrix} A_1 - H_1C_1 & 0 \\ -e^{A_1\tau}H_1C_1 & A_1 - B_1L^* \end{pmatrix} \qquad B_2 = \begin{pmatrix} H_1 \\ e^{A_1\tau}H_1 \end{pmatrix} = (\,(B_2)_1 \quad (B_2)_2\,)$$

$$x_2 = \begin{pmatrix} p \\ \hat{x} \end{pmatrix} \qquad E_2 = \begin{pmatrix} B_1 \\ 0 \end{pmatrix} \qquad C_2 = (\,0 \quad -L^*\,)$$

$$\tilde{B}_2 = (B_2)_1 + A_2(B_2)_2 \qquad \tilde{D}_2 = C_2(B_2)_2$$

The creation of $\tilde{B}_2$ and D_2 is based on the identity:

$$s(sI - A_2)^{-1} = I + (sI - A_2)^{-1}A_2$$

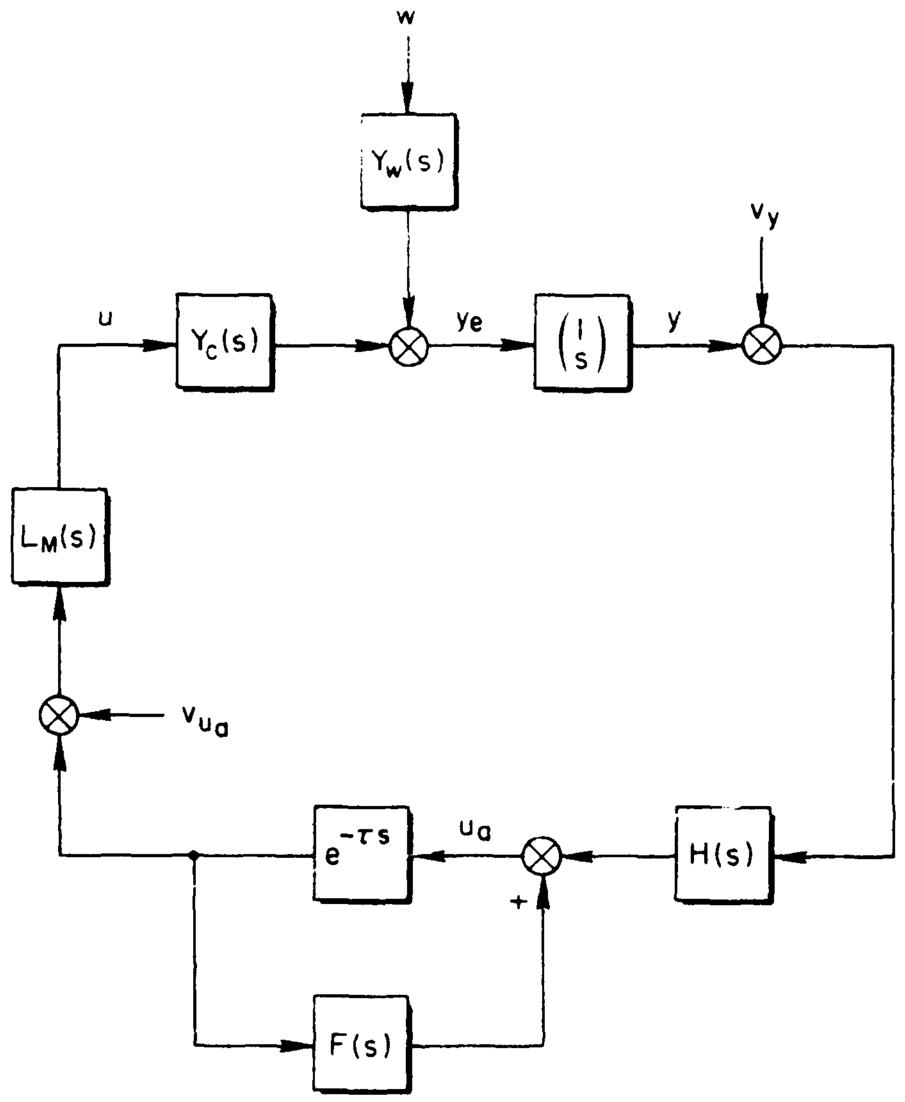


Figure 3: Frequency Domain Version of OCM

Frequency data points can be exactly calculated by using the Laplace transform $e^{-s\tau}$ of the delay. The following steps result in calculation for $Y_p(s)$ and $\Phi_{nn_e}(s)$. Define the following transfer functions, located as shown in Figure 3.

$$\begin{aligned}
H(s) &= (\, H_1(s) \quad H_2(s)\,) = C_2(sI - A_2)^{-1}B_2 \\
H_3(s) &= H_1(s) + sH_2(s) = C_2(sI - A_2)^{-1}\tilde{B}_s + \tilde{D}_2 \\
F(s) &= C_2(sI - A_2)^{-1}E_2 \\
L_m(s) &= (sI - L_2)^{-1}L_2 \\
Y_c(s) &= (C)_1(sI - A)^{-1}B \\
Y_w(s) &= (C)_1(sI - A)^{-1}E \\
& \text{where } C = \begin{pmatrix} (C)_1 \\ (C)_2 \end{pmatrix}
\end{aligned}$$

The transfer function from $y_e(s)$ to $u_a(s)$ is the solution of the implicit equation:

$$u_a(s) = H_3(s)y_\epsilon(s) + F(s)e^{-s\tau}u_a(s)$$

which is:

$$u_a(s) = \left[I - F(s)e^{-s\tau}\right]^{-1} H_3(s)y_\epsilon(s)$$

The pilot model $Y_p(s)$ is the transfer function from $y_\epsilon(s)$ to $u(s)$:

$$\begin{aligned}
Y_p(s) &= L_m(s)G_f(s)H_3(s) \\
& \text{where } G_f(s) = e^{-s\tau}\left[I - F(s)e^{-s\tau}\right]^{-1}
\end{aligned}$$

In order to determine the remnant, use Figure 3 to determine the transfer functions from the noise inputs to the error:

$$y_\epsilon(s) = \underbrace{(I - Y_pY_c)^{-1}\,(\,Y_cL_mG_fH \quad Y_cL_m \quad Y_w\,)}_{G_{cl}(s)} \begin{pmatrix} v_y \\ v_{u_a} \\ w \end{pmatrix}$$

The remant $\Phi_{nn_e}(s)$ is the spectral density resulting from the $v_y(s)$ and $v_{u_a}(s)$ noise sources:

$$\Phi_{nn_e}(s) = G_{cl}(s)\text{diag}\{V_y,\ V_{u_a},\ 0\}G'_{cl}(-s)$$

Up to this point the visual delay of τ seconds has been exactly modeled. It is not, however, possible to obtain a state space description of Y_p unless an approximation is used for the delay. Obtaining a state space realization is quite informative, because by converting to transfer function for $Y_p(s)$, the poles and zeros can be examined. A state space realization of the Pade approximation of a delay is:

$$\begin{aligned}
\dot{x}_p(t) &= A_p x_p(t) + B_p u_a(t) \\
u_p(t) &= C_p x_p(t) + D_p u_a(t) \\
&\quad \text{where } u_p(t) \approx u_a(t - \tau)
\end{aligned}$$

For example, a 2nd order Pade approximation for the single-input-single-output case is:

$$\begin{aligned}
\dot{x}_p(t) &= \begin{pmatrix} 0 & 1 \\ -12/T^2 & -6/T \end{pmatrix} x_p(t) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u_a(t) \\
u_p(t) &= \begin{pmatrix} 0 & -12/T \end{pmatrix} x_p(t) + \begin{pmatrix} 1 \end{pmatrix} u_a(t)
\end{aligned}$$

The 2nd order case usually suffices, though others may of course be tried. Substitute the Pade approximation and group all of the differential equation together to obtain the following. The input y can be compressed as described earlier in order to obtain an equivalent state space system driven just by y_e.

$$\begin{aligned}
\frac{d}{dt}\begin{pmatrix} p \\ \hat{x}_1 \\ x_p \\ u \end{pmatrix} &= \begin{pmatrix} A_1 - H_1 C_1 & -B_1 D_p L^* & B_1 C_p & 0 \\ -e^{-A_1 \tau} H_1 C_1 & A_1 - B_1 L^* & 0 & 0 \\ 0 & -B_p L^* & A_p & 0 \\ 0 & -L_2 D_p L^* & L_2 C_p & -L_2 \end{pmatrix} \begin{pmatrix} p \\ \hat{x}_1 \\ x_p \\ u \end{pmatrix} \\
&\quad + \begin{pmatrix} H_1 \\ e^{A_1 \tau} H_1 \\ 0 \\ 0 \end{pmatrix} (y + v_y) + \begin{pmatrix} 0 \\ 0 \\ 0 \\ L_2 \end{pmatrix} v_{u_a} \\
u &= \begin{pmatrix} 0 & 0 & 0 & I \end{pmatrix} \begin{pmatrix} p \\ \hat{x}_1 \\ x_p \\ u \end{pmatrix}
\end{aligned}$$

Frequency data for $Y_p(s)$ can be obtained directly from the above state space system. In order to obtain the remnant a state space description of the closed loop system must be obtained, but rather than attempt to write it out in terms of all of the elementary matrices, it is best to let a program such as CC do the required algebra. The closed loop system $G_{cl}(s)$ which will be of the form:

$$\begin{aligned} \dot{x}_4 &= A_4 x_4 + B_4 \begin{pmatrix} v_y \\ v_{u_a} \\ w \end{pmatrix} \\ y_e &= C_4 x_4 \end{aligned}$$

As before, the remnant is:

$$\Phi_{nn_e}(s) = G_{cl}(s)\mathrm{diag}\{V_y,\ V_{u_a},\ 0\}G'_{cl}(-s)$$

Several quantities of interest can be obtained from Bode plots of $Y_p(s)$ and $Y_p(s)Y_c(s)$, including:

$$\begin{aligned} \omega_c &= \text{unit magnitude crossover frequency} \\ \phi_{PM} &= \text{phase margin} \\ \tau_e &= (\pi/2 - \phi_{PM})/\omega_c = \text{effective delay} \\ \phi_p &= -\mathrm{Arg}[Y_p(j\omega_c)] + \omega_c\tau_e + \arctan[\omega_c\tau_N] \\ & \quad \text{pilot phase compensation} \end{aligned}$$

These are best obtained by using a cursor to read magnitudes and phases from Bode plots.

3 OCM IMPLEMENTATION

3.1 Overview

The human optimal control model is implemented as a series of macros, two of which call a user defined command[1] The following functions are performed:

1. setup a state space model of the controlled system
2. solve the LQR problem
3. setup the KBF problem
4. compute KBF iterations and performance measures
5. compute state space, transfer function, and/or frequency response models of the human operator

Both the LQR and KBF problems involve iterations, which are automatically controlled by the user defined command. After convergence is obtained, all of the regular features of Program CC are then available for analysis of the results. The types of analysis include frequency responses and simulations. With the ability to obtain transfer function models of the human operator, the types of analysis can be extended to include the root locus, and more importantly to include low order transfer function approximations. It is these low order approximations which clarify the relationship between the optimal control and classical pilot models.

The information for the macros is summarized in Tables 1 through 3. Only the SISO case is implemented.

1. **Table 1**: OCM Macros
2. **Table 2**: Input Parameters
3. **Table 3**: Output Parameters

[1]A *macro* is an indirect command file, whereas a *user defined command* is a compiled BASIC program which can be chained to Program CC

Table 1: OCM Macros

1. OCMYC1, G_i, G_j, $(Y_c)_{all}$

   Create state space $(Y_c)_{all}$ for output driving noise
   $y_e = G_i(s)u + G_j(s)w$

2. OCMYC2, G_i, G_j, $(Y_c)_{all}$

   Create state space $(Y_c)_{all}$ for input driving noise
   $y_e = G_i(s)[u + G_j(s)w]$

3. OCMYC3, G_i, G_j, G_k, $(Y_c)_{all}$

   Create state space $(Y_c)_{all}$ for general driving noise
   $y_e = G_i(s)[G_j(s)u + G_k(s)w]$

4. OCMALL, $(Y_c)_{all}$

   Calls OCMLQR, OCMSETUP, and OCMKBF

5. OCMLQR, $(Y_c)_{all}$, q, g, τ_N, τ_{thresh}

   Automatic iteration of g to achieve desired τ_N

6. OCMSETUP, V_u, τ, ρ_{y_1}, ρ_{y_2}, ρ_{u_a}, V_{y_1}, V_{y_2}, V_{u_a}, T_{y_1}, T_{y_2}, f

   Setup KBF and linear predictor problem

7. OCMKBF,dB_{thresh}

   Automatic iteration of V's for KBF problem

8. OCMPILOT, τ, Pade order

   Create state space models of $Y_p(s)$ and $Y_d(s)$

9. OCMG, G_i, G_j

   Create tfs Y_p and Y_c from state space models

10. OCMFREQ1, ω_{low}, ω_{high}, #pts

    Plot of Y_p, Y_c, Y_pY_c, and Φ_{nn_e} using state space results

11. OCMFREQ2, τ, ω_{low}, ω_{high}, #pts

    Plot of Y_p, Y_c, Y_pY_c, and Φ_{nn_e} using $e^{-s\tau}$

Table 2: Input Parameters

Note: $(Y_c)_{all}$ is a state space quadruple. Do not use names which overlap those used by the macros: $P \rightarrow P_{30}$ and $P_{500} \rightarrow P_{504}$. P_{40} has become a standard name to use for $(Y_c)_{all}$. The parameters G_i, G_j, and G_k are transfer functions, for which any names can be used. All of the remaining parameters are entered as numbers.

$(Y_c)_{all}$ = controlled system and driving noise dynamics, where:

$$\text{input} = \begin{pmatrix} u \\ w \end{pmatrix} \qquad \text{output} = \begin{pmatrix} y_1 \\ sy_1 \end{pmatrix}$$

q, g = quadratic weights, where:

$$J = \lim_{T \to \infty} \mathbf{E} \left\{ \frac{1}{T} \int_0^T \left(q y_e^2 + g \dot{u}^2 \right) dt \right\}$$

τ_N = desired neuro-muscular time constant

τ_{thresh} = desired accuracy

V_w = driving noise intensity

τ = visual delay

ρ_{y_1}, ρ_{y_2}, ρ_{u_a} = noise ratios (entered in dB's)

V_{y_1}, V_{y_2}, V_{u_a} = observation and motor noise intensities

T_1, T_2 = indifference thresholds (0 = no threshold)

f = fractional attention (1 = full attention)

dB_{thresh} = desired dB accuracy of noise ratios

Pade order = order of $e^{-s\tau}$ Pade approximation

ω_{low}, ω_{high}, #pts = Bode plot axis limits

Table 3: Output Parameters, Single-Input Systems

Created by OCMLQR:

$P = (Y_c)_{all}$	$P_3 = g$
$P_1 = \textit{not used}$	$P_4 = L = (\ L_1 \quad L_2\ )$
$P_2 = q$	

Created by OCMSETUP:

$P_5 = L_2$	$P_{10} = EV_{u_c}E'$
$P_6 = L^* = (\ L_2^{-1}L_1 \quad 0\ )$	$P_{11} = \textit{KBF iteration counter}$
$P_7 = L_m = (sI - L_2)^{-1}L_2$	$P_{12} = (\ \rho_{y_1} \quad \rho_{y_2} \quad \rho_{u_1} \quad T_{y_1} \quad T_{y_2} \quad f \quad \tau\ )$
$P_8 = C_1(sI - A_1)^{-1}B_1$	$P_{13} = (\ V_{y_1} \quad V_{y_2} \quad V_{u_a}\ )$
$P_9 = e^{A_1\tau}$	

Created by OCMKBF:

$P_{14} = \Sigma_1$	$P_{19} = X = \hat{E} + \hat{X}$
$P_{15} = H_1$	$P_{20} = (\ \sigma^2_{y_1} \quad \sigma^2_{y_2} \quad \sigma^2_{u_a} \quad \sigma^2_{u} \quad \sigma^2_{\dot{u}}\ )$
$P_{16} = \int_0^{\tau} e^{A_1\sigma}W_1e^{A_1'\sigma}d\sigma$	$P_{21} = \textit{not used}$
$P_{17} = e^{A_1\tau}\Sigma_1e^{A_1'\tau} \quad (\hat{E}=P_{16}+P_{17})$	$P_{22} = \textit{not used}$
$P_{18} = \hat{X}$	$P_{23} = (\ q\sigma^2_{y_1} \quad g\sigma^2_{u_a} \quad J\ )$

Created by OCMPILOT (state space models):

$P_{24} = Y_p$

$P_{25} = Y_{cl} = C_4(sI - A_4)^{-1}B_4 \quad \textit{(closed loop system)}$

Created by OCMFREQ1 and OCMFREQ2 (data files):

yp = human operator	yc = controlled system
$ypyc$ = loop transfer function	phi = remnant

3.2 Using Program CC

Program CC is a command driven computer-aided-control-system-design package. It is written in compiled BASIC and operates under the DOS operating system on the IBM-PC and compatible personal computers. Program CC works with linear systems, either analog or digital, which are modeled with either transfer functions or state space equations. A large number of classical, sampled data, and state space algorithms are implemented, including an extensive amounts of interactive graphics. The user environment in Program CC is robust, friendly, and very powerful.

Program CC uses *macros* and *user-defined-commands* to tailor its use to a particular problem. The SETUP command, as explained here, is used to make sure that the program has access to these features.

Version 4 of Program CC must be used for the optimal control model of the human operator. An introduction to using Program CC, Version 4 is contained in Appendix A. This introduction concentrates on the parts of the program used for the optimal control model. A more extensive introduction is contained in the Tutorial Manual [3]

3.2.1 Setup

Obtain a copy of Program CC, Version 4 and then load it onto the hard disk of an IBM-PC or compatible personal computer. The program consists of about 40 separate modules with the names CC.EXE, CC1.EXE, and so on. Place all of these modules into a subdirectory named CC. Create a subdirectory named CC/DATA to store the data, and a subdirectory named CC/OCM to store the OCM macros. Copy the OCM macros listed in Appendix B into the CC/OCM directory. The OCM macros work only with Program CC, Version 4, and will not run properly with Version 3.

Program CC has the ability to locate data, macros, and modules on different drives. Because of the large number of modules (i.e. overlays) used by Program CC the efficiency of operation is improved by placing some of these in a RAM disk. The following modules which are used by the OCM are listed in order of priority:

CC	root program
CC1	macro processor and utilities
CC25	state space equations
CC31	eigenvalue calculations
CC34	Lyapunov solver
CC2	transfer function display
CC3	polynomial root finder
CC4	transfer function equations

All of the above modules will fit into a RAM disk of 640 Kbytes. It is also possible to put the data used for the OCM into the same RAM disk, in which case it should be expanded by at least 128 Kbytes. For safety, in case of a power failure or a system crash, it is better to place the data in a subdirectory on the hard disk. Data stored on the hard disk can is not lost when Program CC is stopped.

Use the CC/SETUP command to establish the pointers to the above items will be demonstrated after a few more preliminary comments. How to setup a RAM disk depends on your particular hardware configuration and version of DOS. Commands similar to the following should be placed in your CONFIG.SYS file (the files and buffer commands are not related to the RAM disk but are nevertheless a good idea):

```
device=c:\dos\edisk.sys 640 128 512/e
files=20
buffers=32
```

3.2.2 Executing Macros

A macro is an indirect file containing Program CC commands. The commands are listed in a macro exactly as they would be if entered from the keyboard. Macros can be nested and parameters can be inserted. Macros are executed by including the @ symbol before the macro name, for example:

```
CC> @OCMLQR, P40, 1, 1e-3, 5
```

The macros used for the human optimal control model are all written so that they echo the parameters and then pause for a response. Press the **F1** function key to abort, and any other key to continue. The **F1** function key can be pressed at any time to stop execution (though it is not recognized during an overlay swap and sometimes several key strokes are required).

3.2.3 User-Defined-Commands

A *user-defined-command* is a compiled BASIC program which is connected to Program CC during execution. One such user-defined-command is used here, OCM.EXE, which has 2 main functions:

1. Controls the LQR iterations by automatically changing the input weight g.
2. Controls the KBF iterations by automatically updating the observation and motor noise intensities. Also computes the erfc function by means of a piecewise linear approximation, and prints a summary at the end of each KBF iteration.

3.2.4 Example

Start the program from the DOS level by typing CC. Execute the SETUP command as shown below to establish pointers to the data, macros, and modules used by the OCM:

`CC>setup`	SETUP command
`2`	establish pointers to data, macros, and program modules
`2`	use automatic option
`\cc\data`	data location
`\cc\ocm`	macro location
`\cc\data`	more data (not used by the OCM)
`d:`	name of RAM disk
`4`	establish a link to the user defined command
`1`	add a command
`ocm`	name of command
`ocm`	name of BASIC program
`n`	no more commands
`4`	return to main menu
`5`	change the $$SETUP file
`CC>EXIT`	leave the program

Make a back-up copy named SETUP of the file just created. After completing the initialization start Program CC with a DOS batch file such as the following, which copies the SETUP file and the first three of the recommended modules into the D: RAM disk:

```
COPY SETUP $$SETUP
COPY CC.EXE D:
COPY CC1.EXE D:
COPY CC25.EXE D:
CC
```

3.3 The Controlled System

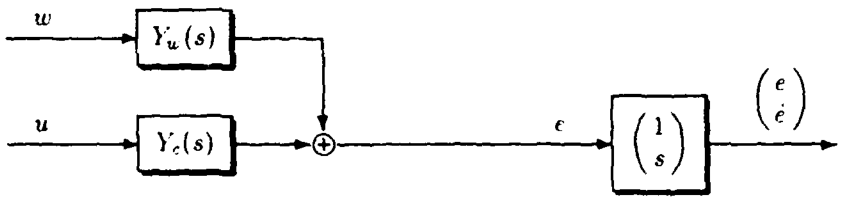
The controlled system, denoted as $(Y_c)_{all}$, includes the controlled element $Y_c(s)$, the noise dynamics $Y_w(s)$, and the interconnection structure. The inputs of $(Y_c)_{all}$ are the operator input u and the external noise source w. The outputs are the error ϵ and the error rate $\dot{\epsilon}$. The structure depends on where the noise is injected, with the three possible choices (for a single-input system) being at the output of $Y_c(s)$, the input, or somewhere between. Use the macros OCMYC1, OCMYC2, and OCMYC3 to create a state space quadruple for $(Y_c)_{all}$ starting from transfer function descriptions of $Y_c(s)$ and $Y_w(s)$. The block diagrams in Figure 4 explain the three cases.

3.4 The Optimal Solution

After creating the model of the controlled system the LQR and KBF optimal control problems must be solved. Each of these is an iterative process. The OCMALL macro defines the following default parameter values and then controls both iterations:

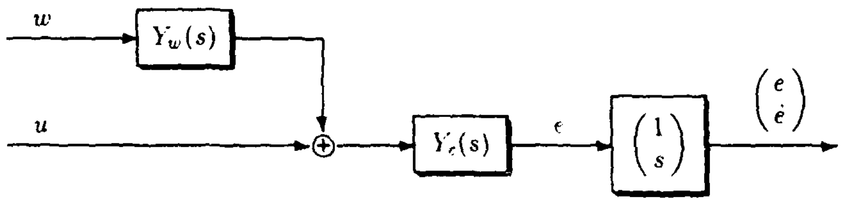
$q = 1$
$\tau_N = .1$
$\tau_{thresh} = .001$
$V_w = 1$
$\tau = .2$
$\rho_{y_1} = \rho_{y_2} = \rho_{u_a} = -20dB$
$T_1 = T_2 = 0$
$f = 1$
$dB_{thresh} = .1$

The OCMALL macro works by calling several lower level macros, two of which call the OCM user defined command (UDC):



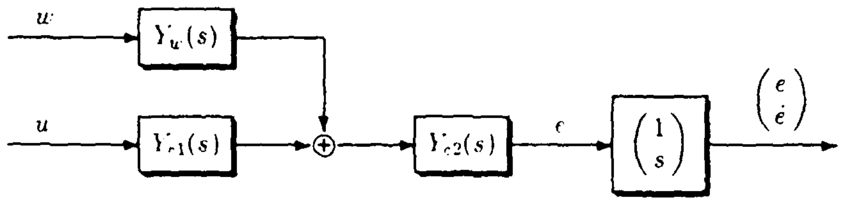
w
$Y_w(s)$
u
$Y_c(s)$
ϵ
$\begin{pmatrix}1\\s\end{pmatrix}$
$\begin{pmatrix}e\\\dot{e}\end{pmatrix}$

(a) Noise Injected at the Output. Created by OCMYC1



w
$Y_w(s)$
u
$Y_c(s)$
ϵ
$\begin{pmatrix}1\\s\end{pmatrix}$
$\begin{pmatrix}e\\\dot{e}\end{pmatrix}$

(b) Noise Injected at the Input. Created by OCMYC2



w
$Y_w(s)$
u
$Y_{c1}(s)$
$Y_{c2}(s)$
ϵ
$\begin{pmatrix}1\\s\end{pmatrix}$
$\begin{pmatrix}e\\\dot{e}\end{pmatrix}$

(c) Noise Injected in the Middle. Created by OCMYC3

Figure 4: Different Configurations for $(Y_c)_{all}$

OCMLQR	solves the LQR problem, calls OCM
OCMSETUP	enters parameters used by OCMKBF
OCMKBF	solves the KBF problem, call OCM
OCMPILOT	computes $Y_p(s)$

The default parameters listed above can be changed by adjusting the inputs to the lower level macros. If these parameters are changed then change the name of OCMALL to something like OCMALL1.

The OCMLQR macro changes the input weight g and repeats the LQR problem until the achieved τ_N is within τ_{thresh} of the desired value. The following ad-hoc method is used: (1) start with an initial guess, (2) increase or decrease by a factor of 10 until the desired τ_N is surrounded, (3) converge using a binary search. After convergence you are presented with the following choices:

1=stop
2=1 more iteration
3=change threshold

Use the first option to stop the LQR iteration and continue to the KBF problem. Use the second option to continue the LQR iteration one more step, and use the third option to set a more precise threshold. The LQR iterations typically go very fast.

The OCMSETUP macro sets up the KBF iterations by storing the required parameters in set locations. The OCMKBF macro controls the KBF iterations, which is a longer and more difficult problem then the LQR iterations. The KBF iterations work by changing V_{y1}, V_{y2}, and V_{u_a}, as described in the technical write-up, until the desired noise ratios are achieved.

At the end of each KBF iteration a summary is presented of the time domain performance measures: optimal cost, the mean square errors, the noise intensities, and the dB noise ratios (see the examples). Any of these can in theory be used as a stopping criteria, however the one which is implemented is for all of the noise ratios to be within dB_{thresh} of their desired values. You can judge the progress of the iteration from the last line of the summary, which give the current value of the maximum noise ratio difference.

A different threshold may be used depending on the computation time, the rate of convergence, and your patience; but more than 1 dB is *not* recommended. Make the change by adjusting the dB_{thresh} parameter in the OCMSETUP macro. The number of required iterations depends on the initial noise intensity estimates, the dynamics of the system (hard to control systems and/or unstable systems tend to require more iterations), the attentional fraction and indifference threshold levels, and the stopping criteria. Anywhere from 3 to more than 50 iterations may be needed. Typically less than 5 iterations are required to reach a noise ratio window of 1 dB, but convergence from that point on can be slow. The best way to significantly reduce the number of iterations is to give good starting values for V_{y1}, V_{y2}, and V_{u_a}. As a practical matter, this can only possible if the same problem or a similar problem has been previously solved. If convergence is not achieved, which sometimes happens, then stop the process using the **F1** function key. Even after stopping it is still possible to continue with further analysis.

When the KBF iteration is completed you are presented with the following choices:

```
1=stop
2=1 more iteration
3=change threshold
4=line printer listing
```

The last option is strongly recommended, which creates a line printer listing of the last iteration summary. Permanent records are always desirable, though in the heat of the moment sometimes forgotten. Use the first option to stop the KBF iterations and continue with the analysis. The second option continues with one more step, and the third option prompts for a smaller threshold.

The OCMPILOT macro creates a state space approximation of $Y_p(s)$. The information needed to compute $Y_p(s)$ is (1) the controlled system $(Y_c)_{all}$, (2) the LQR solution, decomposed into the T_N and L^* matrices, (3) the KBF gains H_1, and (4) the visual delay τ. The state space model created by the OCMPILOT macro is an approximation because a Pade approximation is used in place of an exact delay. A second order approximation is recommended. The parameters for the OCMPILOT macro are the delay and the order of the approximation. The remainder of the information is assumed to be in the locations used by the previous macros. The default name for the state space model of $Y_p(s)$ is P25.

3.5 Analyzing the Results

After executing the OCMALL macro, or its constituent parts, there are several possible directions to proceed with the analysis. The suggestions here are not meant to be exhaustive. Think of the macros mentioned here: OC-MALL, OCMG, OCMFREQ1, and OCMFREQ2, as objects to be changed according to the specific task at hand. It is of course best to change the names if modifications are made.

The time domain performance measures are listed at the end of each KBF iteration. The parameters are stored in matrices as listed in Tables 3. These parameters may be the end point of your analysis, in which case no further work is needed. For example:

1. The optimal cost J can be used to obtain a pilot opinion rating. The value of J is listed in the OCM summary. By referring to Table 3 it is seen that J is also stored in the P_{23} matrix. The definition of J is listed in Table 2; if additional normalization is needed multiply J by the appropriate scale factor.

2. The optimal mean square tracking error $\sigma^2_{y_e}$ can be compared against experimental results. The value is listed in the OCM summary, along with other mean square results, and is also stored in the P_{20} matrix.

The OCMFREQ1 macros use the state space approximations for $Y_p(s)$ and $Y_{cl}(s)$ to compute frequency data files for:

$Y_p(s)$
$Y_c(s)$
$Y_pY_c(s)$
$\Phi_{nn_e}(s)^{1/2}$

The macro ends with a Bode plot created with the PLOT command. The square root of the remnant is computed, as is customary. (The dB scale used in Program CC is $20\log_{10}$. Plotting the square root of the remnant is the same as using a power dB scale of $10\log_{10}$.) The cursor can be used to determine frequency domain parameters such as bandwidth and phase margin.

The OCMFREQ2 macros differ in that they use an exact calculation of $e^{-s\tau}$ as an intermediate step for the same frequency data files as above. I have not yet come across an example where there is any significant difference in the crossover region between the two methods.

Another type of analysis is to obtain a transfer function approximation of $Y_p(s)$, which can then be used to obtain low order approximations. Classical (i.e. structural) pilot model parameters such as pilot lead and effective delay are easiest to obtain using low order models of $Y_p(s)$. After using the

OCMPILOT macro to obtain a state space model for $Y_p(s)$, use the OCMG macro to convert to a transfer function model. The optimal control model of the human operator includes the negative feedback sign with $Y_p(s)$, which is the opposite convention used by the classical human operator model. Multiply the $Y_p(s)$ computed by the OCMG by -1 to switch to the convention used by the classical model. The OCMG macro also computes $Y_c(s)$, in case it is not already available. Multiply $Y_p(s)$ and $Y_c(s)$ together to obtain the loop transfer function, which can then be checked against the K/s crossover rule.

The NEAR command is used in the OCMG macro to cancel poles and zeros of $Y_p(s)$ and $Y_c(s)$. There will be considerable cancellation, and more may be desired. For further cancellation use the NEAR command with larger tolerances. The LFAPPROX (low frequency approximation) command can be used to truncate poles and zeros larger than a threshold frequency, with the option of replacing the effective delay of the truncated modes with a Pade approximation of the delay.

Converting to transfer functions is probably the best way to quickly analyze the results of the OCM, because Program CC's user interaction is faster and more convenient in the transfer function domain. In particular, the plot options can be used for precise bandwidth and phase margin calculations.

3.6 Computation Time

Three factors are important:

Number of states in Y_c
Number of KBF iterations
Micro-processor clock speed

The following operations in each KBF iteration take the most time, each of which is a n^3 operation, where n is the number of states in Y_c:

Riccati equation solution
matrix exponential
Lyapunov equation solution

It is best to use low order closed loop effective models for the controlled system Y_c, and to use only 1st and 2nd order filters for the driving noise. If Y_c has 5 or 6 states, then on an AT with a 8 MHz clock it will take about 50 seconds for each KBF iteration. Due to the n^3 dependence, doubling the number of states requires 8 times as many computations.

The number of inputs is not the dominant factor in computation time (probably a m^2 dependence, where m is the number of inputs). Systems with two or three inputs, however, typically have two or three times as many states, especially if the systems are independent in each axis.

The computation time is proportional to the number of KBF iterations, obviously. How many iterations are required is discussed in Section 3.2.4 Computation time is just as obviously proportional to the micro-processor clock speed. Use your fastest computer.

Overlay and data read times can be very significant if a RAM disk is not used. It is definitely worth the trouble learning how to use a RAM disk.

4 Examples

Duplicate the following OCM examples to gain experience.

4.1 Integrator

The controlled system is a simple integrator in this example taken from [1]. The input of the system is driven by colored noise, the human operator visually determines the error and error rate, and manually controls the input so as to minimize the mean square error. The example has following controlled system and parameters:

$$\begin{aligned}\frac{d}{dt}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &= \begin{pmatrix} -2 & 0 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u + \begin{pmatrix} 1 \\ 0 \end{pmatrix} w \\ \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u\end{aligned}$$

Driving noise: $V_w = 8.8$, resulting in $\mathbf{E}\{x_1^2\} = 2.2$
LQR: $q = 1$, $g = .00017$, resulting in $\tau_N = .08$.
Visual delay: $\tau = .15$
Noise ratios: $\rho_{y_1} = .01$, $\rho_{y_2} = .01$, $\rho_{u_a} = .00316$ (-20,-20,-25 dB)
Indifference thresholds: $t_1 = 0$, $t_2 = 0$
Fractional attention: $f = 1$
Order of Y_c: $n = 2$

The parameters τ_N, τ, and ρ_{u_a} were adjusted by the authors [1] in order to obtain a close match with experimental results. The Program CC implementation agrees with all of the published results except for the remnant frequency response (its not clear whether or not the remnant formulas match).

The state space description of $(Y_c)_{All}$ is provided in the problem statement, and therefore can be directly entered:

```
STATE>p40=(-2,0,0,1; 1,0,1,0; 0,1,0,0; 1,0,1,0)
STATE>p40=CHST(p40,2)
```

The CHST function is used to convert a real matrix into a state space quadruple, in this case with 2 states. It is helpful to compute the same result starting from transfer functions:

```
CC>yc=1/s
CC>yw=1/(s+2)
CC>@OCMYC2,yc,yw,p40
CC>p40=p40(s,(2,1))
```

In this example the noise in injected in the input of the plant, hence the OCMYC2 macro is used to create P_{40}. The odd looking last command switches the order of the 2 states, an optional step which is used to obtain the same state space realization as given in the problem statement.

Now compute the LQR problem. The value of g is already provided, so that no iteration is required:

```
CC>@LQR,p40,1,.00017,.08,.001
```

The τ_N actually achieved is .08074, which falls within the threshold of .001.

There are many parameters required for the KBF problem. List the parameters by running the OCMSETUP macro followed by the **F1** function key, and then include the numbers on the command line. Carefully check before proceeding to the KBF iterations:

```
CC>@OCMSETUP
  F1
CC>@OCMSETUP, 8.8, 0.15, -20, -20, -25, 1,1,1,0,0,1
CC>@OCMKBF,.1
```

Six iterations are required to achieve the desired noise ratios. One more iteration is computed for good measure:

Table 4: KBF Iterations

Iteration	*dB*		
	ρ_{y_1}	ρ_{y_2}	ρ_{u_a}
1	-6.00	-16.23	-15.26
2	-13.47	-15.33	-22.46
3	-17.23	-18.72	-24.32
4	-19.23	-19.68	-24.84
5	-19.81	-19.92	-24.96
6	-19.96	-19.98	-24.99
7	-19.99	-20.00	-25.00

The iteration summary after the 7th iteration is shown in Figure 5. The following noise intensities which were converged to:

$$V_{y1} = .00371 \qquad V_{y2} = .09687 \qquad V_{u_a} = .04815$$

Use this hard achieved information to speed the solution of the problem the next time around:

```
CC>@OCMSETUP, 8.8, 0.15, -20, -20, -25, .00371,
   .09687, .04815 ,0,0,1
CC>@OCMKBF,.1
```

Anytime you want to bring the iteration summary back onto the screen simply call the OCMKBF macro once more. Begin the analysis by noting the time domain performance parameters listed in the summary of the last iteration:

$\sigma^2_{y_1} = .118$
$\sigma^2_{y_2} = 3.08$
$\sigma^2_u = 3.86$
$J = .159$

Run the OCMPILOT macro to obtain a state space model for $Y_p(s)$. Follow this with the GEP command (GEP stands for the Generalized Eigenvalue Problem), which creates a 9th order transfer function model for $Y_p(s)$:

```
CC>@OCMPILOT, 0.15, 2
CC>STATE
STATE>GEP,p24,yp
```

Change the sign of $Y_p(s)$ to conform with the classical convention. Use the NEAR command to cancel poles and zeros with an absolute difference less than 10^{-4}, and then use the LFAPPROX command to replace all of the dynamics greater than 5 rad/sec with a 1st order delay approximation:

```
STATE>cc
CC>yp=-yp
CC>NEAR,yp,yp1,1,1e-4
CC>LFAPPROX,yp1,yp2,5,1
```

Figure 5 shows the $Y_c(s)$ and $Y_u(s)$ transfer functions, as well as the transfer functions for $Y_p(s)$ before and after cancellation. The final low frequency approximation for $Y_p(s)$ is:

$$Y_p(s) = \frac{4.2(s+3.3)}{(s+2.0)}e^{-.13s}$$

```
Iteration # 7
Optimal cost: q,           g,           output,      input rate,  total
              1.0000E+00   1.7000E-04   1.1803E-01   4.1207E-02   1.5923E-01
Performance : E(y_1^2),    E(y_2^2),    E(u_a^2),    E(u^2),      E((du/dt)^2)
              1.1803E-01   3.0834E+00   4.8469E+00   3.8633E+00   2.4240E+02

Old noise intensities (V_y1, V_y2, V_ua):   3.7167E-03  9.6967E-02  4.8176E-02
New noise intensities (V_y1, V_y2, V_ua):   3.7079E-03  9.6869E-02  4.8152E-02
Noise ratios dB (rho_y1, rho_y2, rho_ua): -19.9897    -19.9956    -24.9978
Max noise ratio difference = 1.028061E-02 dB, Threshold = .1 dB


     Controlled element and noise models

               1
     YC(s) = --
               s


               1
     YW(s) = ----
              s+2


     Pilot model and low order approximations

              179.1459( 0)( 2)( 3.252133)( 6.386982)( 12.38519)( 12.56791)
              [-.8660254, 23.09401]
     YP(s) = --------------------------------------------------------------
              ( 0)( 1.990881)( 2)( 6.460116)( 12.37807)( 12.38519)
              [ .3671649, 23.26644]( 42.51991)


               179.1459( 3.252133)( 6.386982)( 12.56791)[-.8660254, 23.09401]
     YP1(s) = ----------------------------------------------------------------
              ( 1.990881)( 6.460116)( 12.37807)[ .3671649, 23.26644]
               ( 42.51991)


              -4.16696( 3.252133)(-15.44067)
     YP2(s) = ------------------------------
              ( 1.990881)( 15.44067)
```

Figure 5: Summary of the Integral OCM Example

This is similar to what a classical (i.e. structural) pilot model would predict, except that the optimal $Y_p(s)$ contains the extra lag term $(s+3.3)/(s+2.0)$. The effect of this term is to increase the low frequency gain, which improves the tracking response to low frequency inputs.

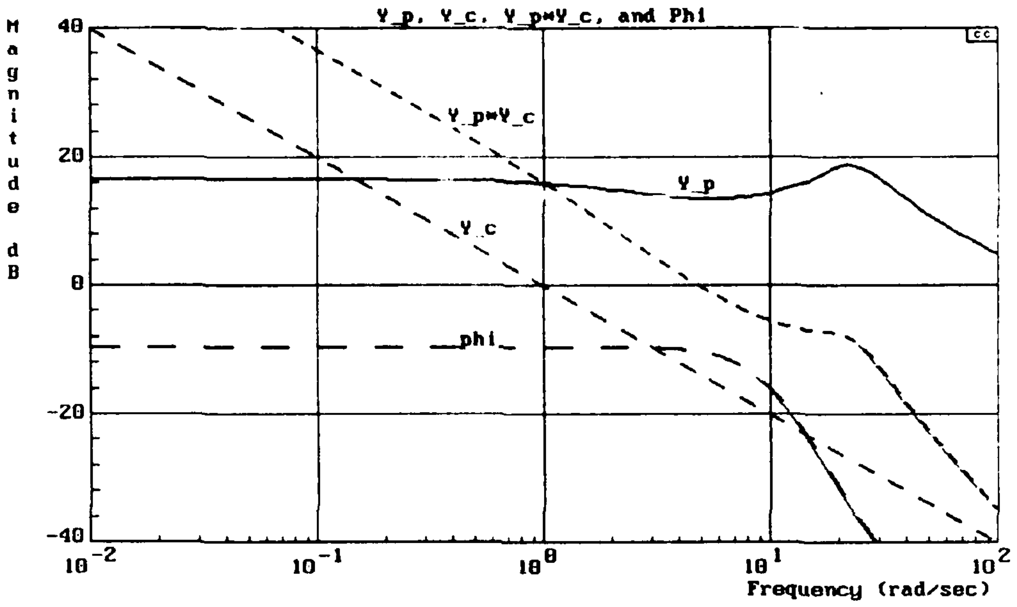
A frequency plot of the important transfer functions is obtained by the following macro call:

```
CC>@OCMFREQ1,.01,100,100
```

The resulting Bode plot shown in Figure 6. The parameters specify 100 points from .01 to 100 rad/sec. The phase of $Y_p(s)$ and $Y_pY_c(s)$ is 180° away from that expected from the classical pilot model convention. Several of the standard frequency domain performance parameters are listed below. (Use the MARGIN and POINT commands to help with some of the following numbers).

$$\begin{aligned} \omega_c &= 4.88 \text{ rad/sec} \\ \phi_{PM} &= 41.9^\circ \\ \tau_e &= \left(\frac{\pi}{2} - \phi_{PM}\frac{\pi}{180}\right)/\omega_c \\ &= .17 \text{ sec} \end{aligned}$$

The commands used to duplicate this example are stored in the EX1.MAC listed in Appendix B.



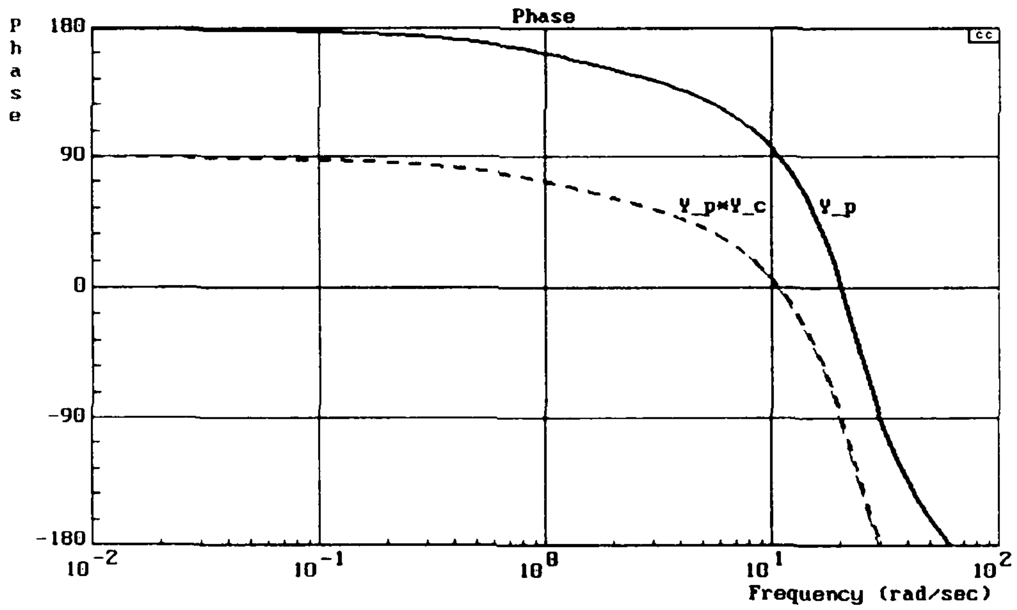


Figure 6: Frequency Responses for Integral OCM Example

4.2 Double Integrator

Another standard human operator tracking problem uses a double integrator. In this example inject the drivi..g noise at the output of the controlled element, and model the noise as unit intensity white noise passed through a 2nd order Butterworth filter with a break at .5 rad/sec. Use defaults for all of the remaining OCM parameters:

Controlled element: $Y_c(s) = 1/s^2$
Driving noise: $Y_w(s) = 1/(4s^2 + 2.83 * s + 1)$
Driving noise: $V_w = 1$
LQR: $q = 1$, $\tau_N = .1$.
Visual delay: $\tau = .2$
Noise ratios: $\rho_{y_1} = .01$, $\rho_{y_2} = .01$, $\rho_{u_a} = .01$ (-20.-20,-20 dB)
Indifference thresholds: $t_1 = 0$, $t_2 = 0$
Fractional attention: $f = 1$

Solve the problem with the following commands:

```
CC>yc=1/s^2
CC>BUTTER,yw,.5,2
CC>@OCMYC1,yc,yw,p40
CC>@OCMALL,p40
```

Obtain transfer function approximations of $Y_p(s)$ with the following additional commands:

```
STATE>GEP,p24,yp
STATE>CC
CC>NEAR,yp,yp1,1,1e-4
CC>LFAPPROX,yp1,yp2,5,1
CC>NEAR,yp2,yp3,1,.2
```

The final result is:

$$Y_p(s) = 2.1(s + 1.3)e^{-.23s}$$

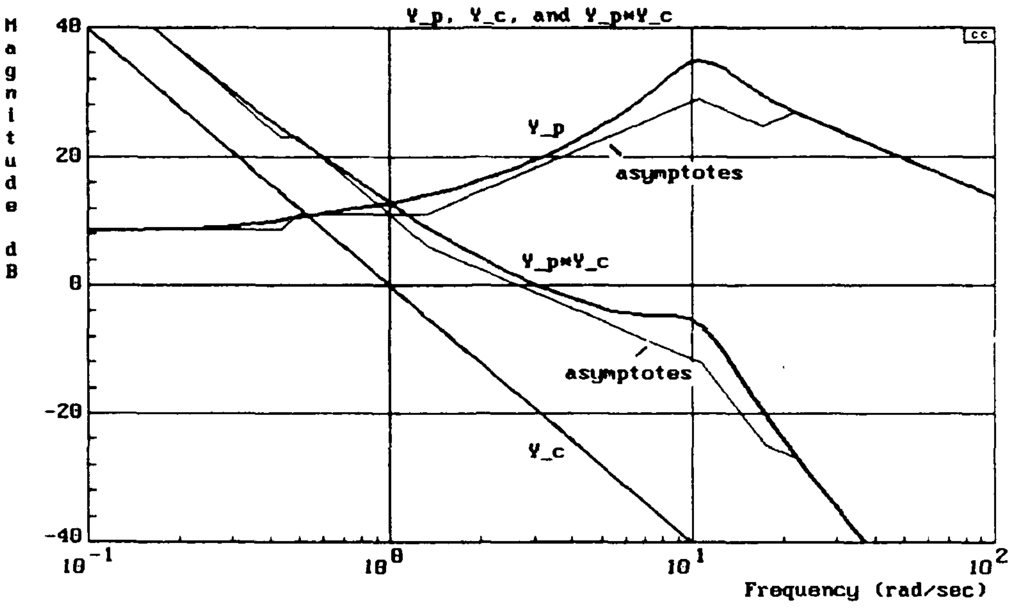
The iteration summary after 16 iterations is shown in Figure 7, together with the transfer function results. The frequency responses are shown in Figure 8. The transfer functions were used to obtain this plot, and hence the straightline asymptotes can be included. The frequency domain performance parameters are listed below:

$$\begin{aligned}\omega_c &= 3.16 \text{ rad/sec}\\ \phi_{PM} &= 26.3^\circ\\ \tau_e &= (\frac{\pi}{2} - \phi_{PM}\frac{\pi}{180})/\omega_c\\ &= .35 \text{ sec}\end{aligned}$$

The commands used to duplicate this example are stored in EX2.MAC. The LQR and KBF problems are started at the parameters which have already been converged to.

```
Iteration # 16
Optimal cost: q,            g,            output,      input rate,  total
              1.0000E+00   6.4053E-05   1.6305E-02   8.0904E-03   2.4395E-02
Performance : E{y_1^2},    E{y_2^2},    E{u_a^2},    E{u^2},      E{(du/dt)^2}
              1.6305E-02   1.1800E-01   2.6566E+00   2.2338E+00   1.2631E+02

Old noise intensities (V_y1, V_y2, V_ua):   5.1638E-04  3.7338E-03  8.4074E-02
New noise intensities (V_y1, V_y2, V_ua):   5.1223E-04  3.7070E-03  8.3460E-02
Noise ratios dB (rho_y1, rho_y2, rho_ua): -19.9650    -19.9688    -19.9682
Max noise ratio difference = 3.499794E-02 dB, Threshold = .1 dB


     Controlled element and noise models

                 1
     YC(s)  =  -----
                 s^2


                 1
     YW(s)  =  --------------
                4s^2 +2.83s+1


     Pilot models and low order approximations

                500( 0)^2 [ .714, .438][ .707, .5]( 1.33)( 3.11)( 10)( 10.1)
                [-.866, 17.3]
     YP(s)  =  -------------------------------------------------------------------
               ( 0)^2 [ .707, .5]^2 ( 3.1)( 9.99)( 10)[ .275, 10.6][ .865, 22.4]

                 500[ .714, .438]( 1.33)( 3.11)( 10.1)[-.866, 17.3]
     YP1(s)  =  ---------------------------------------------------
                [ .707, .5]( 3.1)( 9.99)[ .275, 10.6][ .865, 22.4]

                -2.68[ .714, .438]( 1.33)( 3.11)(-8.69)
     YP2(s)  =  ----------------------------------------
                [ .707, .5]( 3.1)( 8.69)

                -2.05( 1.33)(-8.69)
     YP3(s)  =  -------------------
                ( 8.69)
```

Figure 7: Summary of Double Integrator OCM Example



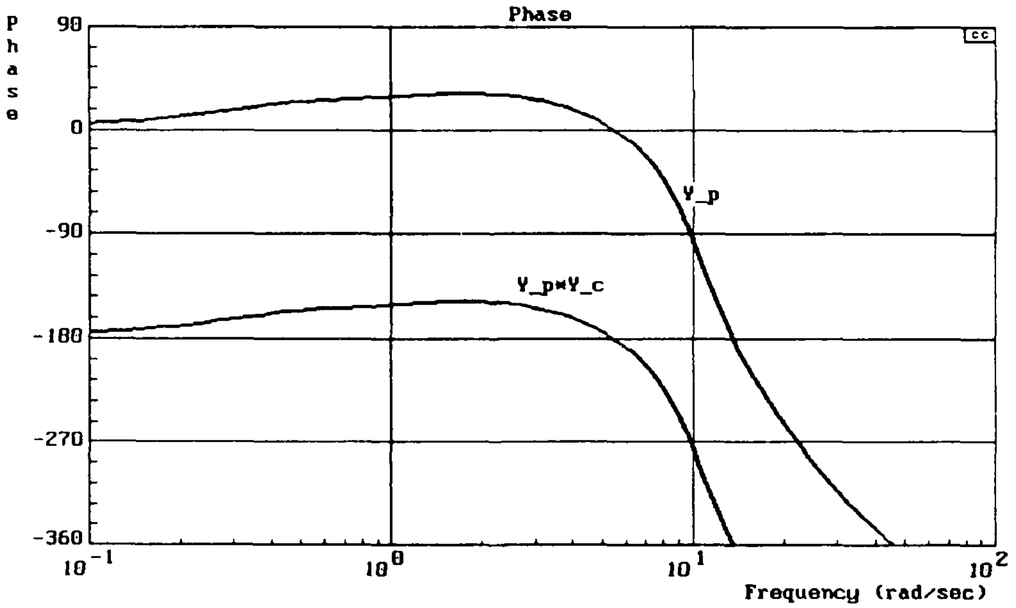


Figure 8: Frequency Responses for Double Integrator OCM Example

4.3 Single Axis Dander Example

Dander [15] has reported experimental findings from single and multi-axis tracking tasks. Dynamically independent single, two-, and three-axis tracking experiments were conducted, and then subjective pilot opinion ratings (PORs) were given for each task using the Cooper-Harper rating procedure. (This procedure yields a POR ranging from 1 (best) to 9 or 10 (worst), which is dominated by the pilot's mental workload required to achieve the performance implied by a given mission phase.) One of the objectives of the original experiment was to predict multi-axis PORs based on single axis results. The best way to make this extension is still a open question, and the Dander data is still the best data base upon which to test out theories.

The first part of this example solves the OCM for a single axis task, and the second part uses the OCM to predict single and multi-axis PORs. The method used to determine the optimal multi-axis cost from the single axis optimal costs is described in Volume 2 of this report. More explanation of the Dander data and POR predictions is provided by McRuer and Schmidt [15] and Volume 2 of this report. Further background on the use of optimal cost for PORs in contained in [13].

The single axis controlled system and driving noise filter are listed below:

$$\begin{aligned} Y_c(s) &= \frac{4(.04)(.9)}{(0)[.7,.25](5)} \\ Y_w(s) &= \frac{.2219}{[.7,.5]} \end{aligned}$$

The driving noise is added to the output of $Y_c(s)$. The mean square output error due to the driving noise is $\sigma_c^2 = .14$. The experimental parameters are listed below, note that nonzero indifference thresholds are used:

V_w	τ	τ_N	ρ_{y1}	ρ_{y2}	ρ_{u_a}	t_1	t_2	f
1	.2	.1	.01	.01	.01	.015	.025	1

Enter the transfer functions, combine them to form the controlled system, and then solve the OCM problem:

```
CC>@OCMYC1,yc,yw,p40
CC>@OCMALL,p40
```

The following time and frequency domain performance parameters have been computed using the OCM solution:

g	$\sigma^2_{y_c}$	$\sigma^2_{\dot{y}_c}$	σ^2_u	$\sigma^2_{u_a}$	$\sigma^2_{\dot{u}}$	J
.00018	.005	.049	.16	.20	9.8	.0067

ω_c	ϕ_{PM}	τ_e
3.2 r/s	37°	.29 sec

See the EX3 macro in Appendix B for more of the commands which duplicate this example. The converged to values of g, V_{y1}, V_{y2}, and V_{u_a} are used in this macro, so that only 1 iteration is needed for each of the LQR and KBF problems. Figure 9 shows the iteration summary and several difference approximations of $Y_p(s)$. Figure 10 shows a Bode plot of the important transfer functions, complete with straightline asymptotes. The low order approximation for the pilot model is:

$$Y_p(s) = \frac{2.6(s+1.4)(s+5.2)}{(s+.9)} e^{-.21s}$$

The classical pilot model is $Y_p = 1.1(s+5)e^{-.25s}$. The lead is used to maintain a K/s like crossover, and the delay is determined as a function of the 3 r/s crossover. The OCM derived Y_p, despite its high order (n+5, where n is the order of Y_c plus the order Y_w), is basically the same around crossover, but differs at lower frequencies by including trim terms, and differs at higher frequencies by including an ill-understood collection of terms which can effectively be grouped into a delay. The much vaunted neuro-muscular mode is cancelled by the OCM solution and does not appear in Y_p. It is nevertheless important to set the neuro-muscular mode as part of the LQR solution, because this method of selecting the quadratic weight determines the closed loop bandwidth.

4.4 Multi-Axis Dander Example

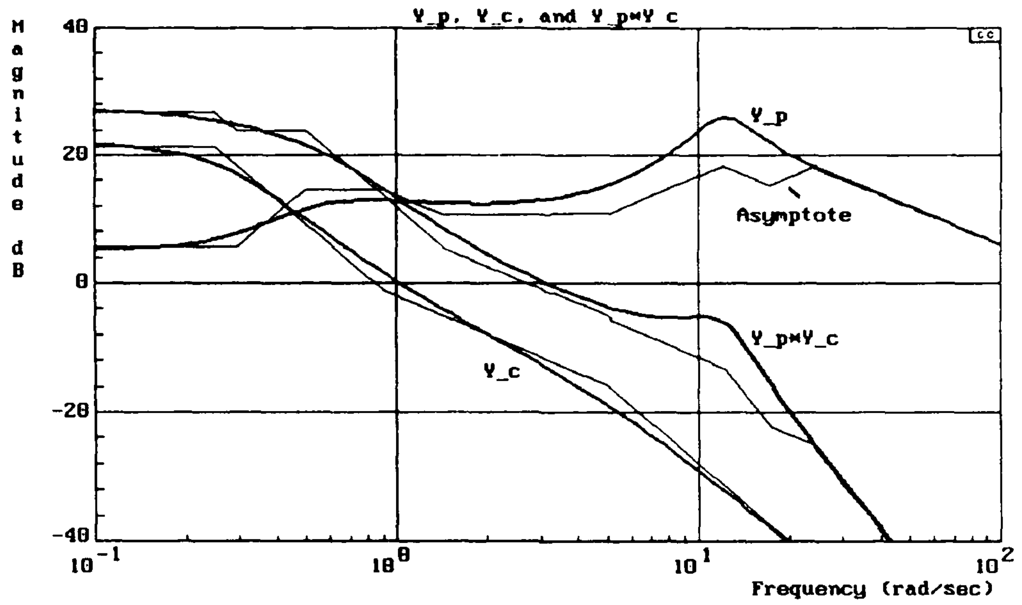
The Dander experiment varied the dynamics in each of 3 axes to create a large number of combinations. Here only a single set is used: $\theta_H \phi_L \beta_H$ in Dander's notation. The OCM is solved for each axis, and then the optimal costs for each axis are combined as demonstrated below to obtain an optimal cost for the multi-axis case. After all of these optimal costs are determined then predictions of PORs are made.

The θ_H axis (axis #1) was considered in the previous subsection. The full attention case for the ϕ_L axis (axis #2) is summarized:

$$Y_c(s) = \frac{0.5(0.1)}{(1.5)[-.84, 0.5]} \qquad Y_w(s) = \frac{13.3}{[.7, .5]}$$

```
Iteration # 1
Optimal cost: q,           g,           output,      input rate,  total
              1.0000E+00   1.8000E-04   4.9850E-03   1.7647E-03   6.7497E-03
Performance : E{y_1^2},    E{y_2^2},    E{u_a^2},    E{u^2},      E{(du/dt)^2}
              4.9850E-03   4.8742E-02   1.9835E-01   1.6504E-01   9.8040E+00

Old noise intensities (V_y1, V_y2, V_ua):   2.7070E-04  2.0260E-03  6.2760E-03
New noise intensities (V_y1, V_y2, V_ua):   2.6840E-04  2.0101E-03  6.2313E-03
Noise ratios dB (rho_y1, rho_y2, rho_ua): -19.9630    -19.9657    -19.9689
Max noise ratio difference = 3.701782E-02 dB, Threshold = .1 dB

     Controlled element and noise models

               4( .04)( .9)
     YC(s)  = ------------------
              ( 0)[ .7, .25]( 5)


              .222
     YW(s)  = ---------
              [ .7, .5]


     Pilot models and low order approximations

               202( 0)( .0299)[ .7, .25][ .71, .296][ .7, .5]( 1.44)( 3.24)( 5)
               ( 5.17)( 10.1)( 10.2)[-.866, 17.3]
     YP(s)  = ---------------------------------------------------------------------
              ( 0)( .04)[ .7, .25][ .7, .5][ .7, .5]( .9)( 3.37)( 5)( 10.1)
               ( 10.1)[ .259, 12.2][ .864, 24.6]


                202( .0299)[ .71, .296]( 1.44)( 3.24)( 5)( 5.17)( 10.2)
                [-.866, 17.3]
     YP1(s)  = ---------------------------------------------------------------
               ( .04)[ .7, .5]( .9)( 3.37)( 5)( 10.1)[ .259, 12.2][ .864, 24.6]


               -.677( .0299)[ .71, .296]( 1.44)( 3.24)( 5)( 5.17)(-9.37)
     YP2(s)  = --------------------------------------------------------
               ( .04)[ .7, .5]( .9)( 3.37)( 5)( 9.37)


               -2.61( 1.44)( 5.17)(-9.37)
     YP3(s)  = -------------------------
               ( .9)( 9.37)
```

Figure 9: Summary of Dander Single Axis Example



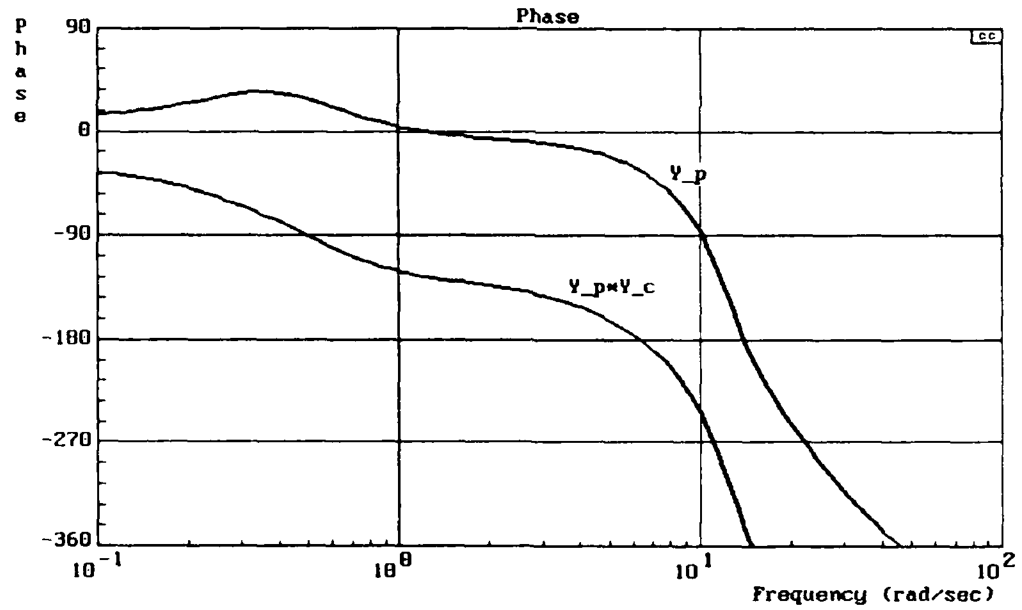


Figure 10: Frequency Responses for Dander Single Axis Example

V_u	t_1	t_2	σ_c^2	g	J
1	.75	1.5	500	10^{-6}	27

and the full attention case for the β_H axis (axis #3) is summarized:

$$Y_c(s) = \frac{10(0.1)}{(3.0)[.5,.5]} \quad Y_w(s) = \frac{.53}{[.7,.5]}$$

V_w	t_1	t_2	σ_c^2	g	J
1	.07	.14	.81	.0013	.05

Each of the single axis OCM problems are solved for several different attentional fractions. How to do this using Program C is now explained. As seen in Table 2, the parameter f is stored in the 1×6 element of the P_{12} matrix. For each different value of f change the P_{12} matrix and repeat the OCMKBF macro. This will start the KBF iterations at the last values of the noise intensities. The iteration summaries for the attentional fraction cases f=1/2, 1/3, 1/4, and 1/5 are listed in Figure 11. The commands used for this attentional fraction survey are listed below (see also the EX3 macro):

```
CC>p12(1,6)=1/2 & @OCMKBF,.1
CC>p12(1,6)=1/3 & @OCMKBF,.1
CC>p12(1,6)=1/4 & @OCMKBF,.1
CC>p12(1,6)=1/4 & @OCMKBF,.1
```

For each axis create a table of optimal costs J versus attentional fractions f. Normalize the costs by dividing by mean square error of the driving noise. It has been empirically determined that a linear relationship exists between J and $1/f$ over a range of attentional fractions, which are for this example:

$$\begin{aligned} J_1/\sigma_{c_1}^2 &= .017/f_1 + .028 && \text{for } f_1 > .1 \\ J_2/\sigma_{c_2}^2 &= .668/f - .212 && \text{for } f_2 > .5 \\ J_3/\sigma_{c_3}^2 &= .018/f_3 + .044 && \text{for } f_3 > .13 \end{aligned}$$

The total optimal cost is the sum of J_1, J_2, and J_3. The following attentional fractions minimize the total cost:

$$\begin{aligned} 1/f_1 &= 1 + \sqrt{a_2/a_1} + \sqrt{a_3/a_1} \\ 1/f_2 &= 1 + \sqrt{a_1/a_2} + \sqrt{a_3/a_2} \\ 1/f_3 &= 1 + \sqrt{a_1/a_3} + \sqrt{a_2/a_3} \end{aligned}$$

```
Iteration # 5     f=1/2
Optimal cost: q,            g,            output,       input rate,  total
              1.0000E+00    1.8000E-04    7.2827E-03    2.0355E-03   9.3182E-03
Performance : E{y_1^2},     E{y_2^2},     E{u_a^2},     E{u^2},      E{(du/dt)^2}
              7.2827E-03    5.8567E-02    2.2780E-01    1.8749E-01   1.1308E+01

Old noise intensities (V_y1, V_y2, V_ua):   6.9377E-04  4.6353E-03  7.0520E-03
New noise intensities (V_y1, V_y2, V_ua):   7.0840E-04  4.7101E-03  7.1565E-03
Noise ratios dB (rho_y1, rho_y2, rho_ua): -20.0907    -20.0695    -20.0639
Max noise ratio difference = 9.065247E-02 dB, Threshold = .1 dB

Iteration # 9     f=1/3
Optimal cost: q,            g,            output,       input rate,  total
              1.0000E+00    1.8000E-04    9.7001E-03    2.2943E-03   1.1994E-02
Performance : E{y_1^2},     E{y_2^2},     E{u_a^2},     E{u^2},      E{(du/dt)^2}
              9.7001E-03    6.7465E-02    2.5446E-01    2.0850E-01   1.2746E+01

Old noise intensities (V_y1, V_y2, V_ua):   1.3042E-03  7.8898E-03  7.8965E-03
New noise intensities (V_y1, V_y2, V_ua):   1.3288E-03  7.9950E-03  7.9940E-03
Noise ratios dB (rho_y1, rho_y2, rho_ua): -20.0813    -20.0575    -20.0533
Max noise ratio difference = 8.126068E-02 dB, Threshold = .1 dB

Iteration # 13    f=1/4
Optimal cost: q,            g,            output,       input rate,  total
              1.0000E+00    1.8000E-04    1.2166E-02    2.5227E-03   1.4689E-02
Performance : E{y_1^2},     E{y_2^2},     E{u_a^2},     E{u^2},      E{(du/dt)^2}
              1.2166E-02    7.5342E-02    2.7794E-01    2.2698E-01   1.4015E+01

Old noise intensities (V_y1, V_y2, V_ua):   2.0933E-03  1.1619E-02  8.6402E-03
New noise intensities (V_y1, V_y2, V_ua):   2.1288E-03  1.1752E-02  8.7318E-03
Noise ratios dB (rho_y1, rho_y2, rho_ua): -20.0731    -20.0491    -20.0458
Max noise ratio difference = 7.305717E-02 dB, Threshold = .1 dB

Iteration # 17    f=1/5
Optimal cost: q,            g,            output,       input rate,  total
              1.0000E+00    1.8000E-04    1.4690E-02    2.7294E-03   1.7419E-02
Performance : E{y_1^2},     E{y_2^2},     E{u_a^2},     E{u^2},      E{(du/dt)^2}
              1.4690E-02    8.2502E-02    2.9919E-01    2.4369E-01   1.5164E+01

Old noise intensities (V_y1, V_y2, V_ua):   3.0646E-03  1.5767E-02  9.3124E-03
New noise intensities (V_y1, V_y2, V_ua):   3.1121E-03  1.5924E-02  9.3992E-03
Noise ratios dB (rho_y1, rho_y2, rho_ua): -20.0668    -20.0430    -20.0403
Max noise ratio difference = .0668335 dB, Threshold = .1 dB
```

Figure 11: Attentional Fraction Survey for Dander Multi-Axis Example

The results for this example are:

$$\begin{aligned} (\, f_1 \quad f_2 \quad f_3 \,) &= (\, .12 \quad .76 \quad .12 \,) \\ J &= 1.03 \end{aligned}$$

Considerably more cost is required to control axis #2, as might be expected from the unstable dynamics in that axis, and as a result considerably more attention is given to axis #2.

The optimal costs just computed can be used to make the following prediction of pilot opional ratings (PORs):

$$\text{HOR} = 5.5 + 3.7\log_{10}\left(\frac{J}{\sigma_c^2\omega_w^2}\right)$$

J is the optimal cost as computed using an error weight $q=1$, σ_c^2 is the mean error of the driving noise, ω_w^2 is the square of the input noise bandwidth, and the numbers 5.5 and 3.7 are experimentally determined using the Dander data and the new STI data in this report. A ± 1 spread in the in the PORs is tolerable. The results for this example are:

axis	$\frac{J}{\sigma_c^2\omega_w^2}$	POR (predicted)	POR (experimental)
1	.045	2.7	2.5–3
2	.456	6.4	5 6.5
3	.062	3.2	3 3.5
all 3	1.03	7.7	7–9

The predictions in this case are very good. All of the 9 different cases of the 3 axis Dander experiments have been similarly analyzed using the OCM implemented in Program CC. Predicted PORs were within ± 1 of the experimental results in 7 out of 9 cases. The POR predictions for the Dander data agrees case-by-case with the analysis reported by McRuer and Schmidt, [16].

A How to Use Program CC, Version 4

Program CC is a command driven computer-aided-control-system-design package. It is written in compiled BASIC and operates under the DOS operating system on the IBM-PC and compatible personal computers. Program CC works with linear systems, either analog or digital, which are modeled with either transfer functions or state space equations. A large number of classical, sampled-data, and state space algorithms are implemented, including an extensive amounts of interactive graphics. The user environment in Program CC is robust, friendly, and very powerful.

This appendix gives an overview of Program CC which is tailored to users of the human optimal control model. It is recommended that you operate the program while reading this introduction. Execute the commands, or at least some of them, when they are introduced.

A.1 The Basics

There are about 300 commands arranged in a hierarchy. Use the manual or the on-line help (command HELP) for a list of the commands and parameters. The hierarchical levels of interest here are:

CC	transfer function commands
STATE	state space commands
DATA	data file commands
MACRO	creating and editing macros

Commands are entered in response to a prompt, for example:

```
CC>HELP
STATE>HELP
```

Commands and parameters are entered in either upper or lower case letters. Blanks are ignored, and only the minimum number of letters to make a command non-ambiguous is needed. Separate the parameters by commas, include them on the command line (expert mode) or let yourself be prompted (novice mode). Include more than one command on a command line by separating them with &.

Branch to different levels of the command hierarchy by typing the name of the command level. Those users familiar with Version 3 should note that the BUILD and MR command levels have been compressed into the CC level. Return to the CC level by typing either QUIT of CC. Leave the program from the CC level by typing either QUIT (prompts with *Are you sure?*) or EXIT (no prompt).

BASIC execution errors are trapped by the program, resulting in an error message and a return to the CC command level. The following commands are built into the program:

Ctrl-NumLock	temporary halt
F1	halt and return to the CC command level
↑ ↓	recall previous commands

Use the **F1** key to abort commands and macros. If waiting for a prompt then follow **F1** with a carriage return. The program does not respond to **F1** if it is pressed during an overlay swap, so sometimes it has to be pressed several times.

All transfer function and state space calculations are computed using double precision arithmetic, which has approximately 18 decimal places of accuracy. The data files are stored in single precision, mainly to save disk space when extreme accuracy is not needed.

Version 4 of Program CC is compatible with the EGA and VGA monitors, but still retains downward compatibility with the CGA. The highest screen mode is automatically determined when the program starts. Monochrome monitors are not recommended because they do not allow graphics, though state space operations (in fact to entire OCM solution) are still possible. Switch screen modes with the SCREEN command. Four color plots can be created on the EGA and VGA. Switch colors with the command COLOR.

Program CC is made of many separate modules, which are chained as needed to execute commands. The modules and the data can be located on more than one disk, as determined by pointers established in the SETUP command. Placing the data and the most used modules in a RAM disk significantly speeds up execution. User defined commands can be included in the list of CC commands, again as established in the SETUP command. When the user defined command is called then the specified program is chained. All of the information established by the SETUP command is stored in the $$SETUP file, and is therefore available each time the program starts. See Section 3.2.1 for information on the best setup for the OCM macros.

There are several more miscellaneous commands of general interest: FILES lists directory files, NAME renames them, and KILL kills them.

CLS clears the screen. TIME and DATE provide their namesakes. CLOCK draws an analog clock. CALCULATOR is an HP-like calculator. ECHO echoes a message and PAUSE pauses until a key is pressed, both of which are useful in macros. SHELL shells to DOS, and EXIT returns to CC. Use DESQVIEW to run multiple copies of CC, or CC and any other set of programs, and switch between the programs with simple DESQVIEW commands.

A.2 Data Types

There are four data types:

Transfer function
State space quadruple
Transfer function matrix
Data file

The names can be six letters or numbers, starting with a letter, followed by a prefix of 3 letters or numbers. For example G, P, YP, $H100$, and $YP.G$. The convention in earlier versions of Program CC, which is still largely maintained in the documentation, is to use the names G_i for transfer functions, P_i for state space quadruples, and H_i for transfer function matrices (the subscript i refers to any positive integer).

ASCII data files are automatically created on disk for each data element. The files are easily recognized because they always begin with $$. The files remain on the disk after the program is finished, do matter how ungracefully, and are therefore available when the program is restarted.

A *transfer function* is a ratio of polynomials. The polynomials can be factored in any desired way. A *state space quadruple* is a packed set of four real matrices which represent a state space differential equation. Real matrices are a subset of state space quadruples, being just the constant term. A *transfer function matrix* is a matrix of transfer functions, usually obtained by converting from a multivariable state space quadruple. Finally, a *data file* is an indexed set of real or complex matrices where the index represents either times or frequencies, and the matrices are ordered row-wise.

The systems are either analog or digital, depending on a flag set with the commands ANALOG and DIGITAL. The program starts with ANALOG as the default. The OCM deals only with analog systems, and therefore the sampled-data features are not emphasize in this introduction.

A.3 The CC Command Level

Transfer functions can be entered either by coefficients, by shorthand notation, or symbolically. For example, to enter:

$$G(s) = \frac{10(s+1)}{s[s^2 + 2(.1)(10)s + (10)^2]}$$

use one of the following commands:

```
CC>GENTER,g, 2,0,10,1,1,1, 2,1,1,0,2,1,2,100
CC>SENTER,g, 2,0,10,1,1, 2,1,0,2,.1,10
CC>g=10*(s+1)/s/(s^2 + 2*.1*10*s + 10^2)
```

Let yourself be prompted for the coefficients until you understand their order. Most users prefer to use the symbolic form. Several commands are available for particular types of transfer functions: BUTTERWORTH, CHEBYSHEV, BESSEL, PADE, LEADLAG, INTEGRATOR, NOTCH, and so on.

Individual coefficients can be changed using the equation interpreter. Its best to precede a change with a display of the transfer function. In the following example, the denominator, 2nd polynomial, 0th order coefficient is changed to 200:

```
CC>DISPLAY,g
CC>g(d,2,0)=200
```

Transfer functions can be displayed in many different forms, as suggested by the command names: DISPLAY, SHORTHAND, SINGLE, and UNITARY, PZF (pole zero form), TCF (time constant form), PFE (partial fraction expansion), and ILT (inverse Laplace transform). In the CC command level entering just the name is equivalent to DISPLAY,G_i. Try, for example:

```
CC>G & SINGLE & PZF & SHO & PFE & ILT
```

The transfer function coefficients are stored in as shown by the DISPLAY command. Change the polynomial factors with the commands CHPZF, CHTCF, CHSINGLE, and CHUNITARY. For example:

```
CC>G & CHSINGLE,G,G1 & G1
```

Transfer functions can be built up from others using the equation interpreter. This powerful facility is used by Program CC in place of block diagrams to compute loop and closed loop transfer functions. For example:

```
CC>G1=G*G1
CC>G3=(G+G1)/(1+G2*(G+G1))
CC>G10=(s+1)*G/s
```

The following example is a simple way to find the closed loop poles with unity feedback:

```
CC>G2=G/(1+G) & PZF,G2
```

The same can be done with the command STABILITY. The range of stable gain can be found with the command ROUTH.

The equation interpreter cancels polynomial factors if the coefficients are linear multiples, but factors are not further broken down to look for cancellations. For example, $(2s + 2)/(s + 1)$ is cancelled but not $(s^2 + 2s + 1)/(s + 1)$. The NEAR command converts to first and second order factors and cancels with ϵ tolerance.

The displays default to single precision. Change this to double or mini (3 decimal places) precision using the FORMAT command. Line printer output of transfer functions is produced by preceding the display style by LP. For example: LPDISPLAY, LPSHORTHAND, LPPZF, and LPTCF. Messages can be included on the output by surrounding the message with quotes, for example:

```
CC>LPSHO,G1,"Closed loop system"
```

Transfer functions can be stored and recalled under arbitrary file names with the commands STORE and RECALL, which is useful for long term saving of special files. Large numbers of transfer functions are more economically saved by using DOS to copy them to a sub-directory.

An e^{-sT} delay can be included with the transfer function using the DELAY command. All transfer functions are multiplied by the same delay, and transfer functions with delays cannot be algebraically combined. It is useful, nevertheless, to use exact delays for frequency response calculations.

Still more transfer function commands are available. ADJOINT computes either $G_i(-s)$ or $G_i(1/z)$. PARTIAL extracts terms from a partial fraction expansion, and SPECTRAL extracts poles in a half plane. MEANSQUARE computes the mean square error. NEAR cancels poles and zeros according to either an absolute or relative criterion. LFAPPROX replaces high frequency poles and zeros with a Pade delay approximation, and HFAPPROX does something similar with low frequency poles and zeros. Use the equation interpreter to extract numerators and denominators and to make substitutions for s:

```
CC>nd=G(n)/G1(d)
CC>gminus=g(-s)
CC>gf=g(f)
```

The following commands require more explanation than is appropriate here: DIOPOLE and DIOLQG are pole-placement and LQG algorithms solved by Diophantine equations. INNER, OUTER, BLASHKY, WIENER, FILTER, and LQG are Wiener-Hopf commands.

A.4 The STATE Command Level

State space quadruples are entered using the command PENTER, which prompts for the number of states, inputs, and outputs; and then prompts for the respective row-wise elements of the A, B, C, and D matrices. All of this can be placed on the command line, for example:

```
STATE>PENTER,P,2,1,1, 1,1,1,1, 2,2, 3,3, 4
```

creates the quadruple:

$$P = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) = \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 1 & 1 & 2 \\ \hline 3 & 3 & 4 \end{array}\right)$$

which represents the state space differential equation:

$$\begin{aligned} \frac{d}{dt}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &= \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 2 \\ 2 \end{pmatrix} u \\ y &= \begin{pmatrix} 3 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 4 \end{pmatrix} u \end{aligned}$$

Real matrices are just the D part of the quadruple. The following command:

```
STATE>PENTER,P1,0,2,3, 1,2,3,4,5,6
```

creates the real matrix:

$$P_1 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

Real matrices can be entered symbolically using the state space equation interpreter. Elements in a row are separated by commas and rows are separated by semicolons. Enter and then display the previous P_1 matrix as follows:

```
STATE>P1=(1,2;3,4;5,6) & P1
```

The equation interpreter does not allow direct entry of state space quadruples, but they can first be entered as a real matrix and then partitioned by changing the number of states:

```
STATE>P=(1,1,2; 1,1,2; 3,3,4) & P=CHST(P,2) & P
```

The command DISPLAY,P_i (or just P_i) is used to display P_i. To display only part of the quadruple use the command P(A). To display the elements in double precision use the FORMAT command. To display just the dimensions use WHAT,P1. To eliminate elements with absolute values less than a threshold use P1=EPS(P1). To obtain a line printer listing use the LPDISPLAY,P1 command, which can contain a message, e.g.:

```
STATE>LPDIS,P15,"KBF gains"
```

Single elements of a matrix or a quadruple can be changed by referring to their position:

```
STATE>P1(1,2)=10
STATE>P(C,1,2)=100
```

This is the easiest type of augmentation. More generally, a real matrix can be augmented at the i,j^{th} position, which overwrites a block of the old matrix and extends the boundaries if necessary.

Real matrices (and most quadruples) can be nested and built up from smaller matrices. For example, to add a row to P_{10}:

```
STATE>P10=(P10; 1,1)
```

The new matrix is built up using augmentation with zero-fill, so there is never a problem with mismatched numbers of rows and columns. To save some typing, for example, the 3×3 identity matrix can be entered in the following way:

```
STATE>p3=(1; 0,1; 0,0,1)
```

Parts of a real matrix or a state space quadruple can be extracted, as shown in the examples:

`STATE>P=P1(1,2)`	single element of P_1
`STATE>P=P1(1)`	1st row of P_1
`STATE>P=P1(,2)`	2nd column of P_1
`STATE>P=P1(1:3,4:5)`	rows and columns of P_1
`STATE>P=P1(A)`	$P_1(A)$ from the P_1 quadruple
`STATE>P=P1(C,,4:5)`	columns 4 and 5 of $P_1(C)$

Several special types of matrices constructed with the following functions: IDEN, DIAG, and RND. Controllable canonical realizations of standard filters are created by the commands BUTTERWORTH, CHEBYSHEV, BESSEL, PADE, LEADLAG, INTEGRATOR, and NOTCH, and so on. Transfer functions can be converted to state space quadruples with the commands CCF, OCF, and DCF; respectively controllable, observable, and diagonal canonical forms. Transfer functions included in state space equations are automatically converted using the CCF realization. Try the following:

```
CC>g=10*(s+1)/(s^3 + 2*s^2 + 3*s+4)
CC>STATE
STATE>p=g
STATE>OCF,g,p1
```

The number sign, #, is used for diagonal augmentation. It is used in the following commands to create 2nd order Pade approximations of diagonal delays:

$$P_{10} = \begin{pmatrix} e^{-.1s} & 0 \\ 0 & e^{-.2s} \end{pmatrix}$$

```
STATE>PADE,P10,.1,2 & PADE,P11,.2,2
STATE>DELAY=(P10#P11)
```

The state space equation interpreter can be used to algebraically combine state space quadruples. Dimensions are checked for validity, and when state space quadruples are used, the appropriate combinations of the A, B, C, and D matrices are automatically computed. The valid operations are:

+	Addition
*	Multiplication
^	Exponentiation
'	Transpose
/	Right division, $\alpha/\beta = \alpha * \beta^{-1}$
\	Left division, $\alpha\backslash\beta = \alpha^{-1} * \beta$
|	Feedback, $\alpha\|\beta = \alpha * (1 + \beta * \alpha)^{-1}$

The levels of precedence are as follows, with operations on the same level computed left to right as they appear in the equation:

```
( )
^
'
* / \ |
+ -
```

An mnemonic for left and right multiplication is that the top of the slash points to the inverted quadruple. The | operation for feedback results in a minimal order state space realization, with α in the feedforward and β in the feedback paths. Identity matrices in the equation are automatically sized. Try, for example, the following:

```
STATE>BUTTER,P,1,4
STATE>P1=P/(I+P)
STATE>P2=P|I
STATE>WHAT,P1 & WHAT,P2
```

Both P_1 and P_2 are closed loop systems with unity feedback, but P_1 is a non-minimal 8th order realization, and P_2 is the much preferred minimal 4th order realization.

There are several ways to invert a matrix (a quadruple is invertible if the D term is invertible):

```
STATE>P1=I/P
STATE>P1=P\I
STATE>P1=P^-1
```

Gaussian elimination (LU decomposition) with scaling and partial pivoting is used to compute the matrix inversion, which takes only order n^3 operations, the same as matrix multiplication. A warning is printed if the determinant is zero. It is more efficient *not* to invert an entire matrix, with the first example below being preferred over the second:

```
STATE>P2=P1\P
STATE>P2=P1\I & P2=P2*P
```

The PACK command packs four real matrices of compatible dimensions into a state space quadruple, and the UNPACK command does the converse. In the following example the P quadruple is unpacked and a second [illegible] of outputs is created which are the derivative of the former. This is only valid if $P(D)=0$. The derivatives are created using the identity:

$$Cs(sI - A)^{-1}B = CB + CA(sI - A)^{-1}B$$

```
STATE>UNPACK,P,P500,P501,P502,P503
STATE>P502=(P502;P502*P500)
STATE>P503=(P503;P502*P501)
STATE>PACK,P500,P501,P502,P503,P
```

A complicated system can be built up using the state space equation interpreter. Series elements are multiplied; parallel elements are either added or augmented, depending on the input and output connections; and feedback paths are closed using the | operation. Use the FEEDBACK command to feedback a subset of outputs to a subset of inputs. In the following example, define P to have 4 inputs and 4 outputs, and define P_1 to be SISO. A single feedback loop is closed around P from the 2nd output to the 3rd input, with P_1 in the feedback path:

```
STATE>P2=P1|((0;0;1;0)*P1*(0,1,0,0))
```

Functions are used in equations. All of the usual trigonometric and exponential functions are available, and when applied to matrices work element-by-element. Other functions which are available include DET, TRACE, and NORM. Most of the complicated operations, however, are computed using commands. The NORM command computes 6 different matrix norms. PSEUDO INVERSE uses the singular value decomposition (SVD) to compute the pseudo inverse of a rectangular matrix. SPACE computes orthonormal basis vectors for the fundamental subspaces. PROJECTION computes projection matrices onto the fundamental subspaces.

Several different matrix decompositions can be computed: EIGENVALUE, SCHUR, HESSENBERG, and SVD. Place a G in front of the first three commands to compute generalized versions. The following example computes the eigenvalue decomposition and the verifies the result. The eigenvalues are stored in a 2 column matrix, which is converted to block diagonal with the DIAG function.

```
STATE>EIG,p,d,x
STATE>p-x*DIAG(d)/x
```

Several different matrix equations can be solved: LYAPUNOV, RICCATI, and SYLVESTOR. Place a D in front of the first two commands to compute discrete versions. Several different methods are available to solve these equations, with the defaults methods being recommended.

The SIMILARITY command computes similarity transformations, either with respect to an input matrix, or with respect to eigenvectors, Schur vectors, or Hessenberg vectors of the system matrix. The CONTROLLABILITY and OBSERVABILITY commands, which are identical, each check for both uncontrollable and unobservable modes; but the algorithm is not foolproof, and if in doubt compute a similarity transformation with respect to the eigenvectors, and check for zero rows of B for uncontrollable modes and zero columns of C for unobservable modes.

Conversion from state space to transfer function matrices is computed using commands FADEEVA and GEP. The former uses the Fadeeva algorithm, which is very fast, but computes polynomial coefficients and is therefore unreliable for large systems ($>$ 6th order depending on the dynamics). The Fadeeva algorithm is used in the following example:

```
STATE>FADEEVA,P,H
STATE>EXTRACT,H,G,1,1 & EXTRACT,H,G1,1,2
```

The transfer function matrix $H(s)$ is created, and then the EXTRACT command is used to extract elements of $H(s)$ into transfer functions. The command EXTRACT,H,G,ALL extracts all of the elements of $H(s)$ rowwise into $G(s)$, $G_1(s)$, and so on. Display transfer function matrices with the same commands used for transfer functions. The denominator, which is common to all of the elements, is only displayed once.

The GEP command uses the generalized eigenvalue problem (hence the name) to compute the poles and zeros, which is slower but numerically more reliable. One problem with the GEP command is that it has trouble with zeros at infinity ($1/s^2$ has 2 zeros at infinity), and for reasons known only to a few tends to put them at $10^{16/m}$, where m is the order of the infinite zeros. The GEP commands allows the user to set a magnitude threshold, above which the zeros are considered infinite. If the state space system is SISO then the conversion can be directly placed into a transfer function, as done in the following example, which has set a threshold of 10^{12} for infinite zeros:

```
STATE>GEP,P1,G1,1e12
```

The POLE PLACEMENT command uses the Ackermann formula for SISO state space pole placement designs. In the following example, the pole placement method is used to compute full state feedback gains (prompts not shown), and then the closed loop system is computed in two different ways:

```
STATE>POLE PLACEMENT,P,P1, ...
STATE>P2=P & P2(A)=P2(A)-P2(b)*P1
STATE>FEEDBACK,4,P,P1,P2
```

Optimal control designs are solved using the commands LQR, KBF, and LQG. Digital versions are solved with the commands DLQR, DKBF, and DLQG. If the intermediate Riccati solution is desired, it must be obtained with a separate call to the RICCATI command. Several different options are available for entering the data, and several choices of computation schemes are available. Its too much to explain here, and one example will have to suffice. The 1st option of the LQR command is used in the following:

```
STATE>LQR,1,P,Q,R,F,1e-4
```

where the parameters are respectively the system, state weight, input weight, and LQR gains. The input weight is multiplied by 10^{-4}. The default computational method using Schur vectors is used to solve the Riccati equation. In the following example, G is the Riccati solution, and F is the LQR gain:

```
STATE>F=P(A) & G=P(B)/(1e-4*R)*P(B)
STATE>RICCATI,F,Q,G,G
STATE>F=(1e-4*R)\P(B)'*G
```

While these several commands are not too complicated, if this sequence is to be repeated then a macro should be constructed, as shown later.

A.5 GRAPHICS

Graphics is one of the best features of Program CC. Frequency and time domain plots can be obtained starting from transfer functions, state space quadruples, or data files.

The plots can be interactively changed, a valuable user-friendly feature. The interaction is accomplished using *plot option blocks*. Figure 12 contains a Bode plot of $G(s) = (s+.2)e^{-.3s}/(s+10)$. The plot option block is located underneath the plot. Press ? (or /) to list the full names of the options, as shown in the bottom of Figure 12, and as listed below:

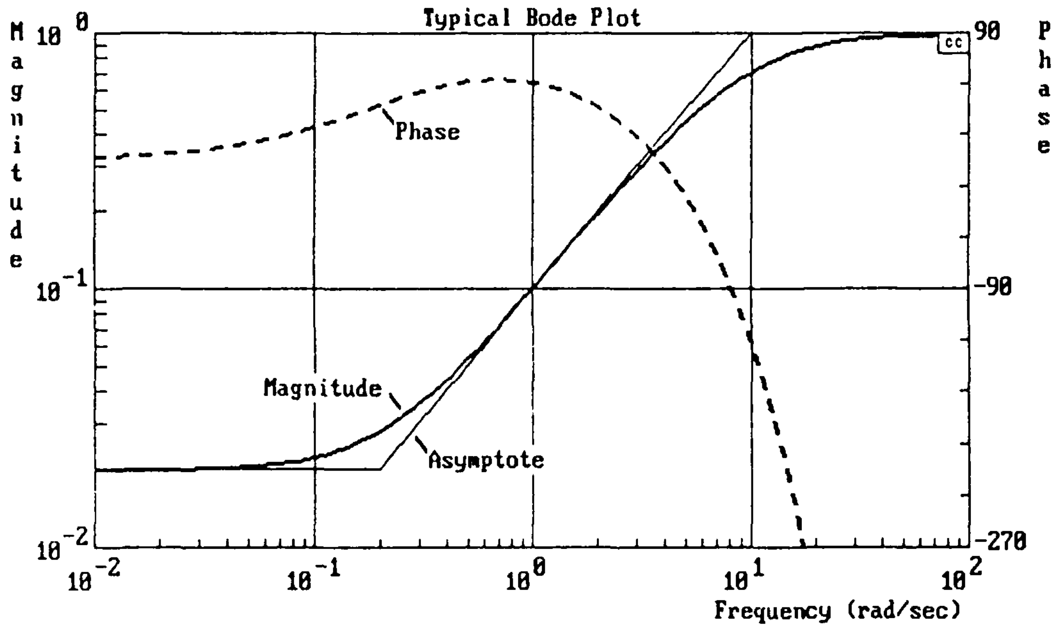
A Overplot with an additional line
B Crosshair cursor

C	Transfer function cursor, includes robustness calculations
D	Change foreground and background options
E	Change axis limits, includes zoom
F	Create data file
G	Use equation interpreter to augment transfer function
H	Hardcopy
I	Automatically fill in with more points
J	Change plot colors
K	Include date and time on title
L	Label the plot
M	Manually fill in with more points
P	Replot
Q	Quit
S	Plot magnitude asymptote
T	Toggle thick and thin lines
W	Change title
X	Pause, for use with macros
Y	Change how phase is calculated
Z	Center plot
?	Help

The following plots are available for transfer functions:

BODE
NICHOLS
NYQUIST
LOG NYQUIST (uses log axes)
ROOT LOCUS (uses gain stepping algorithm)
FASTRL (uses curve tracing algorithm)
SIGGY (Siggy and Bode root loci plots)
TIME (inverse Laplace transform)
DTIME (inverse z-transform)
SIMULATION (simulation of Laplace transform)
DSIMULATION (simulation of z-transform)

The selection and operation of the plot options change depending on the type of plot. It suffices here to use the above BODE example. The axis limits for each plot can either be manually entered or automatically determined.



```
          Options> A B C D E F G H I J K L M P Q S T W X Y Z ?=help

A=addline      E=chg limits  I=fill    M=more pts   T=thick lines  Z=center
B=crosshair    F=data file   J=colors  P=replot     W=title        ?=help
C=tf cursor    G=equation    K=clock   Q=quit       X=pause        tf=G
D=chg options  H=hardcopy    L=label   S=asymptote  Y=chg phase
```

Figure 12: Program CC Plot Including the Plot Option Blocks

The frequency plots work from the FREQ data file, created by the FREQUENCY command. The FREQ file saves time when switching between different frequency files, it allows non-uniform spacing of points, and it saves disk space because the file is overwritten each time the FREQUENCY command is called. To create the Bode plot in Figure 12:

```
CC>G=(s+.2)/(s+10) & DELAY,.3 & SHO,G
CC>FREQ,G,.01,100,100
CC>BODE,3,A
```

As usual, novices enter just the command names and let themselves be prompted. Several different plot options were used to obtain the finished plot.

The TIME and DTIME commands compute functions of time which are then plotted. Options exist for open loop impulse and step responses, closed loop impulse and step responses, and non-causal step responses (e.g. for autocorrelations). Ramp and sinusoidal responses can be computed by first augmenting the transfer function and then plotting an impulse response. The following example plots a closed loop step response and then a closed loop ramp response:

```
CC>TIME,G,1,AUTO  (1=closed loop step)
q
CC>G1=G/(1+G)/S^2
CC>TIME,G1,4,AUTO  (4=open loop impulse)
```

The SIMULATION and DSIMULATION commands compute simulations of transfer functions. The same open and closed loop impulse and step response options are available, but not the non causal option Data files can be used as input, representing either an output from another system or an arbitrary input sequence created by the INPUT command. The SIMULATION command combines a call to the CONVERT command (zero-order-hold equivalence) with a call to the DSIMULATION command. As an alternative:

```
CC>CONVERT,G,G1,3,.1  (Bilinear with T=.1)
DIG>DSIM,G1,2,AUTO  (2=open loop step)
```

When in the digital model the CC command level prompt changes to DIG. Switch back to the analog mode with the ANALOG command.

Time and frequency plots can also be obtained from state space quadruples. The STATE commands SIMULATION and DSIMULATION are used to create time domain data files (unlike the CC commands, then do not directly result in a plot). The simulations can be multivariable, and several different types of inputs are available, including arbitrary data files. The SIMULATION command computes a matrix exponentional to discretize the system and then performs a digital simulation. The name of the time sequence file which is produced is $P_i.Y$.

The FREQUENCY command in the STATE command level is used to compute a frequency data file for a multivariable state space quadruple. The name of the output frequency file is the system name with the file prefix `.G`.

Data files are plotted with the PLOT command. Data files can be created from transfer functions using plot options, by using the DENTER or INPUT command, or as just described by several STATE commands. More parameters are required than for other types of plots. Data files are indexed matrices in general, therefore choices must be made for which rows and columns to plot (ALL for everything). Choices must also be made for what to plot on the horizontal, the left vertical, and optionally the right vertical axes. The axis choices are T(time), W(frequency rad/sec), F(frequency Hz), R(real), I(imaginary), M(magnitude), and P(phase). Each can be prefixed with L($\log_{10}$) or D(dB), with no prefix defaulting to linear. For example:

TR (time plot for simulation data)
LWLM (magnitude Bode plot with $\log_{10}$ scaling)
LWLMP (same including phase)
LWDM, LWDMP, LFDM, WM, FM, WDM (Bode variations)
RI (Nyquist)
PLM, PDM (Nichols)

In the following example a state space simulation is computed and then plotted. The parameters which set up the simulation are not shown here:

```
STATE>SIMULATION,P1, ...
STATE>PLOT,P1,TR,AUTO
```

And a multivariable frequency file is plotted:

```
STATE>FREQ,P, .01,100,60
STATE>PLOT, P.G, LWLM, AUTO
```

The FREQ file created for transfer functions can be directly plotted by the PLOT command, though the plot options are not as convenient:

```
CC>FREQ,G,.01,100,100
CC>PLOT,FREQ,LWDMP,AUTO
```

A.6 The DATA Command Level

Data files are either time sequences or frequency data files as created by Program CC commands. The file structure is very simple, however, and data from any other source can be used. The file structure is:

```
#rows, #columns, type (0=real, 1=complex)
index
real or complex matrix entered row--wise
index
real or complex matrix entered row-wise
:
```

The numbers are stored in ASCII format, and are separated by blanks, commas, semi-colons, or carriage returns. Two carriage returns in a row is interpreted as a zero. The data file is stored on disk with the name *$$name.ext*, and referred to in the DATA command level sans $'s as *name.ext*.

The DATA command level is used to change and algebraically combine data files. The data files are plotted using the PLOT command, which can be called from any command level.

An equation interpreter can be used in the DATA command level to algebraically combine data files, for example:

```
DATA>t=p1.g*p.g
DATA>t=t/(i+t)
```

It is of course best not to let intermediate files proliferate.

Among the most sophisticated commands are those for computing eigenvalues, singular values, and structured singular values of frequency files: respectively EIGENVALUE, SVD, and FMU. For example, to obtain a singular value plot:

```
DATA>SVD, P.G, P.S
DATA>PLOT, P.S, LWDM, ALL, AUTO
```

A.7 Making Macros

There is a great reluctance to using macros. Somehow they must be terribly difficult. They are not — but they do require some learning. The author freely admits reluctance to learn macros from other programs, which for the most part are no more difficult, but always just a little bit different, than those in Program CC.

Follow step by step the creation of the STBODE macro for creating a standardized Bode plot:

```
CC>MACRO
MACRO>ADD
bode,3,0,auto,2,-60,60,6,auto
.
MACRO>STORE,STBODE
MACRO>QUIT
CC>@STBODE
```

Now create a different version of the STBODE macro which first computes the frequency file of an arbitrary transfer function:

```
CC>MACRO
MACRO>ADD
freq,&1,.01,100,60
bode,3,0,auto,2,-60,60,6,auto
.
MACRO>STORE,STBODE
MACRO>QUIT
CC>@STBODE,G50
```

The DAMP macro creates a transfer function with an arbitrary damping ratio. after first echoing a message:

```
CC>MACRO
MACRO>ADD
echo, &1 = transfer function with &2 damping ratio
&1=10*(s+1)/s/(s^2+2*&2*10*s+100)
.
MACRO>STORE,DAMP
MACRO>QUIT
CC>@DAMP,G10,.1
```

The STATE commands RICCATI and LQR are combined in the following macro. The parameters are echoed, and then the PAUSE command gives the user to abort with the **F1** function key.

```
CC>MACRO
MACRO>ADD
echo, The LQR1 Macro
```

```
echo, &1 = system
echo, &2 = state weight Q
echo, &3 = input weight R
echo, &4 = input parameter rho
echo, &5 = Riccati solution P
echo, &6 = LQR gains F
echo, Hit F1 to abort, any other key to continue
pause
state
&6=&1(a) & &5=&1(b)/(&4*&3)*&1(b)
riccati,&6,&2,&5,&5
&6=(&4*&3)\&1(b)'*&5
echo, End of LQR1 macro
.
MACRO>STORE,LQR1
MACRO>QUIT
CC>@LQR1,P,P1,P2,1e-4,P3,P4
```

Okay, so that one wasn't so easy.

The MACRO command level is a mini-text editor. So far you've seen the ADD and STORE commands. There are several more commands to do things like LIST, SEARCH, DELETE, and REPLACE; but rather than explain it here just use the HELP command.

Everything that gets entered goes into HISTORY file (disk file $$HIS); which can be recalled in the MACRO command level, edited, and played back as a demonstration. Or if you want to create a standard plot but cannot remember the parameters, then create the plot once using prompts and build a macro which includes the prompts. Plot labels get stored along with the cursor m ments, which show up in the HISTORY file as weird symbols. Here's a simple example, with no editing, of a demonstration macro:

```
CC>MACRO
MACRO>HISTORY
MACRO>STORE,DEMO
MACRO>QUIT
CC>@DEMO
```

If your disk space is limited and the HISTORY file is getting too long, delete it by leaving CC and returning, or more elegantly by using the TRUNCATE command:

```
CC>MACRO
MACRO>TRUNCATE,0
MACRO>QUIT
```

Okay, that's enough. Macros aren't so tough. I won't pressure you into learning more.

B Macro Listings

OCMYC1

```
echo, OCMYC1 Macro
echo, Purpose: Create (Y_c)_all with driving noise at output
echo, (Y_c)_all=(1;s)*(Gi,Gj)
echo,
echo, SISO controlled system (Gi) = &1
echo, SISO noise filter (Gj) = &2
echo, MIMO (Y_c)_all (Pi) = &3
echo,
echo, Hit function key F1 to abort, any other key to continue
pause
state
&3=(&1,&2)
&3(c,0,1)=&3(c)*(&3(a),&3(b))
quit
echo, Finished OCMYC1
```

OCMYC2

```
echo, OCMYC2 Macro
echo, Purpose: Create (Y_c)_all with driving noise at input
echo, (Y_c)_all=(1;s)*Gi*(1,Gj)
echo,
echo, SISO controlled system (Gi) = &1
echo, SISO noise filter (Gj) = &2
echo, MIMO (Y_c)_all (Pi) = &3
echo,
echo, Hit function key F1 to abort, any other key to continue
pause
state
&3=&1*(1,&2)
&3(c,0,1)=&3(c)*(&3(a),&3(b))
quit
echo, Finished OCMYC2
```

OCMYC3

```
echo, OCMYC3 Macro
echo, Purpose: Create (Y_c)_all with general driving noise
echo, (Y_c)_all=(1;s)*Gi*(Gj,Gk)
echo,
echo, SISO controlled system before noise (Gi) = &1
echo, SISO controlled system after noise (Gj) = &2
echo, SISO noise filter (Gk) = &3
echo, MIMO (Y_c)_all (Pi) = &4
echo,
echo, Hit function key F1 to abort, any other key to continue
pause
state
&4=&1*(&2,&3)
&4(c,0,1)=&4(c)*(&4(a),&4(b))
quit
echo, Finished OCMYC3
```

OCMALL

```
echo, OCMALL Macro
echo, Purpose: Complete human OCM problem, using defaults
echo,
echo, (Y_c)_all = &1
echo,
echo, Replaces OCMLQR, OCMSETUP, and OCMKBF
echo, F1 to abort, any other key to continue
pause
@ocmlqr,&1,1,1e-4,.1,.001
@ocmsetup,1,.2,-20,-20,-20,1e-2,1e-2,1,0,0,1
@ocmkbf,.1
@ocmpilot,.2,2
```

```
echo, Finished OCMALL
```

OCMLQR

```
echo, OCMLQR Macro
echo, Purpose: Adjust LQR weights for tau_N
echo,
echo, Y_c = &1
echo, q = &2
echo, g = &3
echo, tau_N = &4
echo, thresh = &5
echo,
echo, Use Function Key F1 to abort, any other key to continue
pause
state
p=&1
p2=&2
p3=&3
quit
ocm,lqr,&4,&5
echo, Finished OCMLQR Macro
echo, Next use OCMSETUP and OCMKBF Macros
```

OCMSETUP

```
echo, OCMSETUP Macro
echo, Purpose: Setup KBF iterations for Human OCM
echo,
echo, intensity of driving noise: V_w = &1
echo, visual delay: tau = &2
echo, y_1 noise ratio (dB): rho_(y_1) = &3
echo, y_2 noise ratio (dB): rho_(y_2) = &4
```

```
echo, u_a noise ratio (dB): rho_(u_a) = &5
echo, initial y_1 noise intensity (non-zero): V_(y_1) = &6
echo, initial y_2 noise intensity (non-zero): V_(y_2) = &7
echo, initial u_a noise intensity (zero okay): V_(u_a) = &8
echo, y_1 indifference threshold: T_1 = &9
echo, y_2 indifference threshold: T_2 = &10
echo, axis fractional attention: f = &11
echo,
echo, Execute after OCMLQR
echo, Hit Function Key F1 to abort, any other key to continue
pause
state
echo,-----------------------Augmentation
p5=p4(1,cdim(p4))
p6=(p4(1,1:cdim(p4)-1)/p5,0)
p7=chst((-p5,p5;1,0),1)
echo,-----------------------Setup KBF problem
p8=p(,1)*p7
p9=p8(a) & p9=chst(p9,rdim(p9))
expon,p9,p9,&2 & p9=chst(p9,0) & analog
p10=p(b,,2)*&1*p(b,,2)'
p11=0
p12=(&3,&4,&5,&9,&10,&11,&2)
p13=(&6,&7,&8)
quit
echo, Finished OCMSETUP Macro
echo, Next use OCMKBF Macro
```

OCMKBF

```
echo, OCMKBF Macro
echo, Purpose: KBF/linear predictor iterations for OCM
echo,
echo, noise ratio threshold (dB) = &1
echo,
echo, Execute after OCMLQR and OCMSETUP
```

```
echo, Hit Function Key F1 to abort, any other key to continue
pause
ocm,kbf,&1
echo, Finished OCMKBF Macro
```

OCMPILOT

```
echo, OCMPILOT Macro
echo, Purpose: Create SISO state space models of Y_p and Y_cl
echo,
echo, Visual delay: tau = &1
echo, Order of Pade approximation of delay = &2
echo,
echo, Execute after OCMLQR, OCMSETUP, and OCMKBF
echo, Hit function key F1 to abort, any other key to continue
pause
state
echo,-------------------Augmentation preliminaries
pade,p500,&1,&2
p501=(p8(a)-p15*p8(c),-p8(b)*p500(d)*p6,p8(b)*p500(c))
p502=(-p9*p15*p8(c),(p8(a)-p8(b)*p6;-p500(b)*p6,p500(a)))
p501=(p501;p502)
p502=(p15;p9*p15;0*p500(b))
p503=(0*p6,-p500(d)*p6,p500(c))
pack,p501,p502,p503,,p500
echo,--------------------Pilot
p24=p500
p24(b)=(p24(b,,1)+p24(a)*p24(b,,2);p24(c)*p24(b,,2))
p24=p7*p24(,1)
echo,---------------------Closed loop system
p25=(p*(p7*(p500,1)#1))|-(1,0;0,1;0;0)
p25=p25(1)
kill,$$p500 & kill,$$p501 & kill,$$p502 & kill,$$p503
quit
echo, End of OCMPILOT Macro
echo, Continue analysis using OCMG and OCMFREQ1
```

OCMG

```
echo, OCMG Macro
echo, Purpose: Convert Y_p and Y_c from state space to tfs
echo,
echo, Y_p: G_i = &1
echo, Y_c: G_j = &2
echo,
echo, Execute after OCMPILOT
echo, Hit function key F1 to abort, any other key to continue
pause
state
p500=p(1,1)
fadeeva,p500,&2
kill,$$p500
gep,p24,&1
quit
near,&1,&1,1,1e-4
near,&2,&2,1,1e-4
echo, End of OCMG Macro
echo, Continue analysis with regular CC commands
```

OCMFREQ1

```
echo, OCMFREQ1 Macro
echo, Purpose: Create freq plots of Y_p, Y_c, Y_p*Y_c, Phi
echo, Assumes Y_p=P24 and Y_cl=P25 computed using OCMPILOT
echo,
echo, low freq = &1
echo, high freq = &2
echo, # points (log spaced) = &3
echo,
echo, Execute after OCMPILOT
```

```
echo, Hit function key F1 to abort, any other key to continue
pause
state
echo,-------------------compute yp, yc, and yp*yc
frequency,p24,&1,&2,&3
kill,$$yp & rename,$$p24.g,$$yp
p500=p(1,1)
frequency,p500,&1,&2,&3
kill,$$yc & rename,$$p500.g,$$yc
data
ypyc=yp*yc
echo,-------------------compute phi
state
p500=p25(,1:3)
p501=diag(p13)
frequency,p500,&1,&2,&3
kill,$$phi & rename,$$p500.g,$$phi
data
phi=phi*p501*phi'
phi=phi/p20(1,1)
phi=phi^.5
phi=real(phi)
kill,$$p500 & kill,$$p501
echo,-------------------plot result
plot,yp,lwdm,all,auto,-60,60,6,"Y_p, Y_c, Y_p*Y_c, and Phi"
A
YC
ALL
2
A
YPYC
ALL
3
A
PHI
ALL
4
```

OCMFREQ2

```
echo, OCMFREQ2 Macro
echo, Purpose: Create freq plot of Y_p, Y_c, Y_p*Y_c, and Phi
echo, Uses exact calculation of delay
echo,
echo, tau = &1
echo, low freq = &2
echo, high freq = &3
echo, # points = &4
echo,
echo, Execute after OCMLQR, OCMSETUP, and OCMKBF
echo, Hit function key F1 to abort, any other key to continue
pause
state
echo,-------------------Compute Y_c
p500=p(1,1)
frequency,p500,&2,&3,&4
kill,$$yc & rename,$$p500.g,$$yc
echo,-------------------Compute Y_p
p500=(p8(a)-p15*p8(c);-p9*p15*p8(c),p8(a)-p8(b)*p6)
p501=(p15,p8(b);p9*p15)
p501=(p501,p501(,1)+p500*p501(,2))
p502=(0*p6,-p6)
p503=(0,0,0,p502*p501(,2))
pack,p500,p501,p502,p503,p500
frequency,p500,&2,&3,&4
kill,$$t & rename,$$p500.g,$$t
int,p500 & dig,&1 & frequency,p500,&2,&3,&4 & analog
kill,$$delay & rename,$$p500.g,$$delay
frequency,p7,&2,&3,&4
kill,$$nm & rename,$$p7.g,$$nm
p501=diag(p13)
' t = (H,F,H_3), delay = exp(-s*tau), nm = neuro-muscular
data
yp=nm*delay|-t(,3)*t(,4)
ypyc=yp*yc
echo,-------------------Compute Phi
```

```
delay=delay|-t(,3)
delay=delay*t(,1:2)
delay=(delay,1)
'
phi=yc|-yp
phi=phi*nm
phi=phi*delay
'
phi=phi*p501*phi'
phi=phi/p20(1,1)
phi=phi^.5
phi=real(phi)
kill,$$p500 & kill,$$p501 & kill,$$p502 & kill,$$p503
kill,$$delay & kill,$$nm
echo,-------------------plot result
plot,yp,lwdm,all,auto,-60,60,6,"Y_p, Y_c, Y_p*Y_c, and Phi"
A
YC
ALL
2
A
YPYC
ALL
3
A
PHI
ALL
4
```

EX1

```
echo, Enter controlled system (2 different ways)
state
p40=(-2,0,0,1; 1,0,1,0; 0,1,0,0; 1,0,1,0)
cc
yc=1/s
yw=1/(s+2)
```

```
@ocmyc2,yc,yw,p40
state
p40=p40(s,(2,1))
p40
echo, Solve the OCM
@ocmlqr,p40,1,.00017,.08,.001
@ocmsetup,8.8,.15,-20,-20,-25,.00371,.09687,.04815,0,0,1
@ocmkbf,.1
@ocmpilot,.15,2
state
gep,p24,yp
yp=-yp
echo, Low order tf approximations
near,yp,yp1,1,1e-4
lfapprox,yp1,yp2,5,1
```

EX2

```
echo, Enter controlled system
yc=1/s^2
butter,yw,.5,2
@ocmyc1,yc,yw,p40
echo, Solve the OCM
@ocmlqr,p40,1,6.4053e-5,.1,.001
@ocmsetup,1,.2,-20,-20,-20,5.1223e-4,3.7070e-3,8.3460e-2,0,0,1
@ocmkbf,.1
Echo, Low order tf approximations
state
gep,p24,yp
cc
yp=-yp
near,yp,yp1,1,1e-4
lfa,yp1,yp2,5,1
near,yp2,yp3,1,.2
```

EX3

```
echo, Enter controlled system
senter,yc,3,0,4,1,.04,1,.9,3,1,0,2,.7,.25,1,5
senter,yw,1,0,.2219,1,2,.7,.5
@ocmyc1,yc,yw,p40
echo, Solve the OCM, full attention case
@ocmlqr,p40,1,.00018,.1,.001
@ocmsetup,1,.2,-20,-20,-20,.0002707,.002026,.006276,.015,.025,1
@ocmkbf,.1
Echo, Low order tf approximations
@ocmpilot,.2,2
state
gep,p24,yp
cc
yp=-yp
near,yp,yp1,1,1e-4
lfa,yp1,yp2,8,1
near,yp2,yp3,1,.3
Echo, Fractional attention cases
p12(1,6)=1/2 & @ocmkbf,.1
p12(1,6)=1/3 & @ocmkbf,.1
p12(1,6)=1/4 & @ocmkbf,.1
p12(1,6)=1/5 & @ocmkbf,.1
p12(1,6)=1/6 & @ocmkbf,.1
```

References

[1] D.L. Kleinman, S. Baron, and W.H. Levison, *An Optimal Control Model of Human Response, Part I: Theory and Validation*, **Automatica**, Vol. 6, pp. 357–369, 1970.

[2] K.M. Doyle and W.C. Hoffman, *Pilot Modeling for Manned Simulation, Volume II: Program User's Manual (PIREP)*, AFFDL-TR-76-124 Volume II, Aerospace Systems, Inc., Dec. 1976.

[3] P.M. Thompson, **Program CC Version 4 Tutorial and Reference Manual**, Systems Technology, Inc., 1988.

[4] S. Baron, D.L. Kleinman, D.C. Miller, W.H. Levison and J.I. Elkind, *Application of Optimal Control Theory to the Prediction of Human Performance in a Complex Task*, AFFDL-TR- 69-81, Bolt, Beranek and Newman, Inc., March 1970.

[5] D.L. Kleinman, *Optimal Control of Linear Systems with Time-Delay and Observation Noise*, **IEEE Trans. Auto. Control**, Oct. 1969.

[6] W.H. Levison, J.I. Elkind, and J.L. Ward, *Studies of Multivariable Manual Control Systems: A Model for Task Interference*, NASA CR-1746, May 1971.

[7] W.C. Hoffman, R.E. Curry, D.L. Kleinman and W.M. Hollister, *Display/Control Requirements for VTOL Aircraft*, ASI-TR-75-26, Aug. 1975.

[8] W.H. Levison, S. Baron and Junker, *Modeling the Effects of Environmental Factors on Human Control and Information Processing*, AMRL-TR-76-74.

[9] R.E. Curry, W.C. Hoffman and L.R. Young, *Pilot Modeling for Manned Simulation, Volume I*, AFFDL-TR-76-124 Volume I, Aerospace Systems, Inc., Dec. 1976.

[10] R.A. Hess, *Analysis of Aircraft Attitude Control Systems Prone to Pilot-Induced Oscillations*, **J. Guidance**, Vol. 7, No. 1, 1984.

[11] R.O. Anderson, *A New Approach to the Specification and Evaluation of Flying Qualities*, WPAFB, Ohio, AFFDL-TR-69-120, 1970.

[12] J.D. Dillow and G.P. Picha, *Application of the Optimal Pilot Model to the Analysis of Aircraft Handling Qualities*, WPAFB, Ohio, AFIT-TR-75-4, 1975.

[13] R.A. Hess, *Prediction of Pilot Opinion Ratings Using an Optimal Pilot Model*, **Human Factors**, 19(5), pp. 459–475, 1977.

[14] D.K. Schmidt, *Pilot/Vehicle Analysis of Multi- Axis Tasks*, Notes, Purdue Univ. and Systems Technology, Inc., Sept. 1986.

[15] V. Dander, *An Evaluation of Four Methods for Converting Single Axis Pilot Ratings to Multi-Axis Pilot Ratings Using Fixed Base Simulation Data*, M.S. Thesis, AFIT, GE/EE/62-4, Dec. 1962.

[16] D. McRuer and D.K. Schmidt, *Pilot-Vehicle Analysis of Multi-Axis Tasks*, AIAA Guidance, Navigation and Control Conference, Monterey, CA, Aug. 1987.

☆ U.S.Government Printing Office 1989 748-056 24134